Rechnen in Maschinensprache

Praktische Tips und Tricks braucht jeder Programmierer. Wir zeigen Ihnen, wie in Maschinensprache gerechnet wird. Mit vielen Beispielen erklären wir Ihnen ausführlich alles Notwendige - Schritt für Schritt.

In diesem Artikel werden die wichtigsten Routinen des Basic-Interpreters zum Umgang mit Zahlen erläutert, zur Division, Multiplikation und so weiter, und die Nutzung dieser Routinen in Assemblerprogrammen. Sollten Sie bereits von der sogenannten »Fließkommadarstellung« gehört haben, die im Zusammenhang mit diesen Routinen ständig benutzt wird und ziemlich komplex ist: Durch diesen Artikel sollen Sie vor allem den praktischen Umgang mit den Arithmetikroutinen lernen. Zahlenformate werden nur behandelt, soweit Sie für das Verständnis unbedingt erforderlich sind. Wenn Sie mehr über die Fließkommadarstellung wissen wollen, empfehle ich Ihnen den Kurs »Assembler ist keine Alchimie« von H.Ponnath (Sonderheft 8/85, Assembler für Einsteiger und Fortgeschrittene).

Wer benötigt die Arithmetikroutinen?

Die Kenntnis dieser Routinen ist vor allem für zwei Typen von Programmierern interessant. Zum einen für jene »Freaks«, die extreme Anforderungen an Kürze und Geschwindigkeit von Programmen stellen und diese daher vollständig in Maschinensprache beziehungsweise Assembler schreiben.

Solange in reinen Maschinenprogrammen die meistverwendeten Integerzahlen nur addiert oder subtrahiert werden müssen, hält sich der Programmieraufwand noch in Grenzen. Die meisten Assemblerprogrammierer werden mir jedoch zustimmen, daß bereits die Multiplikation und Division von Integerzahlen einen recht hohen Aufwand erfordert und es unumgänglich wird, eigene Unterprogramme zur Multiplikation, Division oder zur Ausgabe von Integerzahlen auf dem Bildschirm zu schreiben.

Beim Rechnen mit Integerzahlen können diese Unterprogramme recht sinnvoll sein. Wenn Sie in Ihren Maschinenprogrammen jedoch auch mit Realzahlen arbeiten, ist es unsinnig, auch dafür eigene Routinen zu erstellen. Alle (!) Routinen, die zum Umgang mit Realzahlen benötigt werden, befinden sich bereits in unserem Computer und zwar im Basic-Interpreter, der sie bei der Programmabarbeitung ständig benötigt.

Die zweite Gruppe sind Programmierer, deren Programme vorwiegend in Basic geschrieben werden, die jedoch Unterroutinen in Maschinensprache als Ersatz für besonders zeitkritische Basic-Programmteile einsetzen. Wenn in solchen Routinen gerechnet, zum Beispiel multipliziert werden muß, ist es unsinnig, nach Basic zurückzukehren und die benötigte Rechnung dort vorzunehmen, um anschließend das errechnete Ergebnis in Maschinensprache weiterzuverarbeiten.

Durch die Übergabe der Zahlen an das Basic-Programm (Lesen mit: PEEK(Zahl1), PEEK(Zahl2)) und die Rückgabe des Ergebnisses an die Maschinenroutine mit POKE(Adresse), (Ergebnis) würde derart viel Zeit vergehen, daß der Geschwindigkeitsvorteil der Maschinenroutine in vielen Fällen zunichte gemacht wird. Die Berechnungen müssen in solchen Fällen ebenfalls im Maschinenprogramm durchgeführt werden.

Bei gemischten Basic- und Assemblerprogrammen stellt sich oft das Problem der Parameterübergabe und des Zugriffs auf eine Basic-Variable von einem Maschinenprogramm aus. Ein Beispiel: Eine Maschinenroutine soll die Werte eines Integer- oder Realarrays aufsummieren. Das erste Problem besteht in der Parameterübergabe: Wie wird der Maschinenroutine durch das Basic-Programm der Name des Arrays übermittelt, dessen Werte aufsummiert werden sollen? Das zweite Problem besteht darin, daß die Summierungsroutine »herausfinden« muß, wo im Computerspeicher das Array durch den Basic-Interpreter abgelegt wurde.

Nach der Besprechung der Arithmetikroutinen werde ich erläutern, welche Möglichkeiten es zur Parameterübergabe an ein Maschinenprogramm gibt, und wie der Zugriff auf Basic-Variablen von Maschinensprache aus möglich ist.

Gleich und doch nicht gleich

Kompatibilitätsprobleme zwischen C16, C64 und C128

Wie Sie vielleicht wissen, enthält das Betriebssystem von Commodore-Computern eine sogenannte »Kernel-Sprungtabelle«, eine Liste der wichtigsten Routinen des Betriebssystems und der jeweiligen Einsprungadresse. Diese Sprungtabelle gewährleistet eine problemlose Übertragbarkeit von Assemblerprogrammen, die Betriebssystemroutinen verwenden, zwischen verschiedenen (Commodore-) Computern. Unabhängig davon, ob Sie einen C16, einen C64 oder einen C128 besitzen, können Sie mit der Befehlsfolge:

```
CLC
LDX #5 (Zeile)
LDY #5 (Spalte)
JSR $FFF0 (PLOT, setzt den Cursor)
```

den Cursor immer direkt auf Spalte fünf von Zeile fünf setzen. Die Kompatibilität wird dadurch hergestellt, daß erst nach dem Einsprung in $FFF0 der Sprung zur eigentlichen Routine erfolgt. Zum Beispiel befindet sich beim C64 an Adresse $FFF0 der Befehl »JMP $E50A«, ein Sprungbefehl mit der C64-spezifischen Adresse der Routine PLOT.

Diese Sprungtabelle enthält jedoch keinerlei Routinen des Basic-Interpreters, die im folgenden fast ausschließlich ver-

wendet werden. Die folgenden Programme beziehen sich alle auf den C 64. Am Ende dieses Kapitels befindet sich eine Übersicht der verwendeten Routinen und Adressen. Soweit sie mir bekannt sind, finden Sie in dieser Tabelle auch die entsprechenden Adressen beim C 16 und C 128, so daß der Großteil der vorgestellten Listings auch für den C 16 oder den C 128 umgeschrieben werden kann, da die Funktionsweise der Arithmetikroutinen trotz unterschiedlicher Adressen auf allen drei Computern identisch ist. Alle (!) verwendeten Routinen besitzen eine Unterprogrammform, enden also mit einem »RTS« und können problemlos mit »JSR $....« aufgerufen werden.

Die Redaktion ist dankbar für jeden Tip eines Lesers, der Lücken in dieser Tabelle füllen kann, oder aber gar weitere Arithmetikroutinen in seinem Computer entdeckt hat.

Zahlendarstellung im Integer- und Fließkommaformat

In Assemblerprogrammen wird vorwiegend mit Integerzahlen gearbeitet, das heißt mit ganzen Zahlen. Wie groß die darzustellenden Zahlen sein können, hängt davon ab, wie viele Bytes zur Darstellung verwendet werden. Bei Verwendung eines Bytes für eine Integerzahl ergibt sich ein Wertebereich von 0 bis 255, das heißt insgesamt 256 = 2^8 verschiedene ganze Zahlen können dargestellt werden.

Bei Verwendung von zwei Byte = Word = 2^{16} können bereits 65536 verschiedene Zahlen dargestellt werden (0 bis 65535). In fast allen Assemblerprogrammen ist die Verwendung von Zwei-Byte-Zahlen und der sich daraus ergebende Wertebereich völlig ausreichend.

Dieser Wertebereich kann übrigens auch negative Zahlen umfassen, wenn die sogenannte »vorzeichenbehaftete Darstellung« verwendet wird. Bei dieser Art der Darstellung dient das oberste Bit zur Darstellung des Vorzeichens.

```
1000 0000 0000 0000
```

Ist es gelöscht, handelt es sich um eine positive, ist es gesetzt, um eine negative Zahl. Da das als Vorzeichen verwendete Bit zur Angabe der absoluten Größe entfällt, beträgt die größte mit zwei Byte darstellbare vorzeichenbehaftete Zahl 32767. Da der Wertebereich auch auf negative Zahlen ausgeweitet wird, können alle Zahlen zwischen -32768 und +32767 dargestellt werden.

```
..+ 32767 = 7fff = 0111 1111 1111 1111
  + 32766 = 7ffe = 0111 1111 1111 1110
  + 32765 = 7ffd = 0111 1111 1111 1101
    ...     ...          ...
  +     1 = 0001 = 0000 0000 0000 0001
        0 = 0000 = 0000 0000 0000 0000
  -     1 = ffff = 1111 1111 1111 1111
    ...     ...          ...
  - 32766 = 8002 = 1000 0000 0000 0010
  - 32767 = 8001 = 1000 0000 0000 0001
..- 32768 = 8000 = 1000 0000 0000 0000
```

Die zweite Art der Zahlendarstellung, das Fließkommaformat, erlaubt die Darstellung beliebiger Realzahlen, der Wertebereich ist daher extrem groß im Vergleich mit der üblichen Zwei-Byte-Integerdarstellung. Alle (!) Zahlen, auch Integerzahlen, wandelt der Basic-Interpreter übrigens zuerst in Fließkommazahlen, bevor arithmetische Operationen mit ihnen ausgeführt werden.

Dennoch ist es keineswegs notwendig, daß Sie wissen, wie Zahlen im Fließkommaformat dargestellt werden. Zur Umwandlung der in Maschinenprogrammen üblichen Integerzahlen ins Fließkommaformat existieren Routinen, die noch von mir besprochen werden.

Vorläufig genügt es, wenn Sie sich merken, daß der Basic-Interpreter Fließkommazahlen in Form von fünf Byte im Speicher ablegt, wobei ein Byte für den »Exponenten« und vier Byte für die »Mantisse« verwendet werden. Das Vorzeichen der Zahl wird mit dem obersten Bit eines der vier Mantissenbyte dargestellt. Ist das Bit gelöscht, ist die Zahl positiv, bei gesetztem Bit ist sie negativ.

Die Fließkomma-Akkumulatoren

Die arithmetischen Operationen des Basic-Interpreters finden in den beiden sogenannten »Fließkomma-Akkumulatoren« statt, in FAC # 1 und FAC # 2, die meistens FAC und ARG genannt werden. Es handelt sich dabei um zwei Speicherbereiche in der Zeropage, die zur Aufnahme der beiden Zahlen verwendet werden, die miteinander verknüpft werden sollen.

Aufbau von FAC und ARG (C 64)

	FAC	ARG
Exponent	$61	$69
Mantisse 1	$62	$6A
Mantisse 2	$63	$6B
Mantisse 3	$64	$6C
Mantisse 4	$65	$6D
Vorzeichen	$66	$6E

Adressenvergleich

	C 64	C 16	C 128
FAC	$61-$66	$61-$66	$63-$68
ARG	$69-$6E	$69-$6E	$6A-$6F

Der Aufbau von FAC und ARG ist bei allen drei Computern identisch.

Wie die Tabelle zeigt, sind beim C 16 und dem C 64 sogar die Adressen identisch. Vielleicht wundern Sie sich über die zusätzliche Speicherstelle für das Vorzeichenbyte, da ich vor kurzem noch behauptet habe, das Vorzeichen sei in einem Bit der Mantisse »versteckt«: FAC und ARG haben eine etwas verschwenderischere Art der Fließkommadarstellung.

Der Rechner im Computer

Im gesamten Speicher bis auf FAC und ARG werden Fließkommazahlen tatsächlich im kompakten Fünf-Byte-Format abgelegt. Wenn nun eine Fließkommazahl, zum Beispiel die Basic-Variable V1, zur Vorbereitung einer Rechnung nach FAC oder ARG kopiert wird, wird die komplette Mantisse1 nach $66 beziehungsweise nach $6E in das Vorzeichenbyte kopiert, und das oberste Bit von Mantisse1 immer (!) gesetzt (mit ORA # $80), bevor dieses Byte an seine Position im FAC transportiert wird.

Wenn umgekehrt eine Fließkommazahl in FAC oder ARG an eine beliebige Speicheradresse transportiert werden soll, wird das oberste Bit von Mantisse1 aus dem Vorzeichenbyte rekonstruiert, bevor der Transport erfolgt (FAC: LDA $66:ORA # $7F:AND $62).

Diesen Unterschied zwischen dem kompakten Fünf-Byte-Format, das im Speicher verwendet wird, und dem speziellen FAC/ARG-Format müssen Sie unbedingt berücksichtigen, wenn Sie Fließkommazahlen »per Hand« zwischen FAC oder ARG und dem restlichen Speicher hin- und hertransportieren wollen. Wenn ein solches Transferproblem in Ihren Programmen auftritt, und Sie die dafür vorhandenen Routinen nicht verwenden können, weil die entsprechenden Adressen für Ihren Computer in der Tabelle fehlen, müssen Sie beim Kopieren auf dieses Vorzeichenbit beziehungsweise -byte unbedingt achten.

Wenn zwei Zahlen miteinander verknüpft werden sollen (zum Beispiel 10 * 3 oder 10/3), müssen beide Zahlen zuerst ins Fließkommaformat konvertiert werden. Anschließend müssen die Zahlen nach FAC und ARG übertragen werden. Im letzten Schritt wird die jeweilige Arithmetikroutine aufgerufen, zum Beispiel die Multiplikations- oder die Divisionsroutine. Bei allen Arten der Verknüpfung befindet sich das Ergebnis anschließend im FAC.

Bevor nun verschiedene Assemblerprogramme folgen, will ich kurz die wichtigsten Eigenarten des von mir verwendeten MAE-Assemblers beschreiben:

MAE	Hypra-Ass	Bedeutung
.BA $c000	.BA	Startadresse eines Programms
.OS	.OB	Anweisung, Objektcode zu generieren
.FAC.de $61	.EQ FAC=$61	Zuweisung eines Wertes
.LDA #L,FAC	LDA #<(....)	Low-Byte-Wert eines Wertes
.LDA #H,FAC	LDA #<(....)	High-byte-Wert eines Wertes
.BY	.BY	Byte

Ausgabe von Fließkommazahlen

Für alle folgenden Beispielprogramme benötigen wir eine Möglichkeit, das aus der Verknüpfung von zwei Zahlen resultierende Ergebnis möglichst problemlos überprüfen zu können, möglichst eine Ausgabe des Ergebnisses direkt auf dem Bildschirm.

Der Basic-Interpreter stellt zwei Routinen zur Verfügung, die kombiniert zur Ausgabe einer Zahl (die sich im Fließkommaformat im FAC befinden muß) auf dem Bildschirm führen. Die betreffende Zahl wird nach dem Aufruf der beiden Routinen an der momentanen Cursorposition ausgegeben.

FAC-Inhalt in ASCII-String wandeln ($BDDD)

Bevor diese Routine (FACSTR) aufgerufen wird, muß dafür gesorgt werden, daß sich die auszugebende Zahl im Fließkommaformat im FAC befindet. Beim Aufruf der Routine müssen keine weiteren Parameter übergeben werden, das heißt Akku, X- und Y-Register können beliebige Inhalte besitzen. Die Routine wandelt die Fließkommazahl in einen ASCII-String um, der mit Hilfe einer weiteren Routine auf dem Bildschirm ausgegeben werden kann.

Achtung! Die Fließkommazahlen im FAC werden durch die Umwandlung zerstört. Wenn Sie nach der Ausgabe mit diesen Zahlen weiterrechnen wollen, müssen Sie vor Aufruf von FACSTR den Inhalt des FAC in einen freien Speicherbereich kopieren.

ASCII-String ausgeben ($AB1E)

Auch diese Routine (STROUT) benötigt keine weiteren Übergabeparameter. Der Aufruf mit »JSR $AB1E« genügt zur Ausgabe der zuvor in einen String umgewandelten Zahl.

Zahlenausgabe auf dem Bildschirm

```
Eingabe:
        FAC         = Wert
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

jsr FACSTR          ; $bddd
jsr STROUT          ; $ab1e

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Inhalt zerstört
```

Konstante im Fließkommaformat nach FAC übertragen ($BBA2)

Um die Wirkung beider Routinen zu demonstrieren, benötigen wir eine Fließkommazahl im FAC. Beim C 64 befindet sich an Adresse $BBA2 die Routine KONFAC, die eine beliebige Konstante im Fließkommaformat in den FAC überträgt.

Diese Routine konvertiert die Zahl zugleich vom kompakten Fünf-Byte-Speicherformat in das Sechs-Byte-Format der Fließkommaakkus (ein zusätzliches Byte für das Vorzeichen und Setzen des obersten Bits von Mantisse1). Die entsprechenden Adressen beim C 16 und C 128 sind mir leider nicht bekannt. Wenn Sie diese Routine in Ihrem Computer nicht finden sollten, können Sie sie jedoch problemlos durch eine Schleife ersetzen, mit der Sie die fünf Byte »per Hand« in den FAC kopieren (vergessen Sie nicht, Mantisse1 in das Vorzeichenbyte zu kopieren und das oberste Bit dieser Mantisse im FAC zu setzen!).

Vor dem Aufruf dieser Routine müssen Akku und Y-Register mit der Adresse geladen werden, an der sich die betreffende Konstante befindet (Akku=Low-Byte/Y=High-Byte).

FAC mit Konstante laden

```
Eingabe:
        Akku        = Low-Byte-Adresse
        X-Register  = unbenutzt
        Y-Register  = High-Byte-Adresse
lda #<(Wert)        ; Low-Byte-Adresse
ldy #>(Wert)        ; High-Byte-Adresse
jsr KONFAC          ; $BBA2
Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Konstante
```

Bildschirmausgabe einer Fließkommazahl

Im Programm »Zahlenausgabe« (Listing 1) wird die Zahl PI (3,14...) verwendet, deren Fließkommadarstellung lautet: $82 $49 $0F $DA $A1.

Das Programm befindet sich an Adresse $C000, die Zahl PI am Programmende. Nach dem Aufruf des Programms mit »SYS 49152« wird der Akku mit dem Low- und das X-Register mit dem High-Byte der Adresse geladen, an der sich die Zahl befindet. Anschließend wird die Routine KONFAC aufgerufen, die die Konstante in den FAC überträgt.

Im FAC befindet sich nun die Zahl PI in der benötigten Form. Durch den Aufruf von FACSTR wird der FAC-Inhalt in einen ASCII-String gewandelt, der mit der Routine STROUT auf dem Bildschirm an der momentanen Cursorposition ausgegeben wird (3,14159265).

FAC als Konstante speichern

FACKON ($BBD4) ist die Umkehrung von KONFAC und kopiert den Inhalt des FAC an eine übergebene Adresse, wobei gleichzeitig die Konvertierung in das gepackte Format vorgenommen wird. Im Gegensatz zu KONFAC wird die Adresse im X- und Y-Register übergeben (X=Low-Byte/Y=High-Byte).

FAC als Konstante speichern

```
Eingabe:
        FAC         = Konstante
        Akku        = unbenutzt
        X-Register  = Low-Byte-Adresse
        Y-Register  = High-Byte-Adresse
jsr FACKON          ; $bbd4
Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
```

Konvertierungsroutinen

In der Praxis werden Sie häufig nicht mit Fließkomma-, sondern mit Integerzahlen arbeiten. Um Arithmetikoperationen mit zwei Integerzahlen durchzuführen, müssen diese jedoch zuvor in das Fließkommaformat gewandelt werden. Der Basic-Interpreter besitzt glücklicherweise sowohl Routinen zur Wandlung von Integerzahlen ins Fließkommaformat als auch zur Wandlung von Fließkommazahlen in das Integerformat.

Die wichtigsten dieser »Konvertierungsroutinen« werde ich im folgenden besprechen und zwar:

1. Ein-Byte-Integer nach Fließkomma wandeln.
2. Zwei-Byte-Integer nach Fließkomma wandeln.
3. Fließkomma nach Integer wandeln.

Ein-Byte-Integer (positiv) nach Fließkomma ($B3A2)

Meist wird bei der Verwendung von Integerzahlen die positive Darstellung verwendet, das heißt, daß der Ein-Byte-Wert als positive ganze Zahl zwischen 0 und 255 aufgefaßt wird. Mit der Routine EINPOS wird eine solche Zahl in das Fließkommaformat umgewandelt. Die entsprechende Fließkommazahl befindet sich nach dem Aufruf im FAC. Vor dem Aufruf der Routine muß das Y-Register mit der zu konvertierenden Zahl geladen werden.

Das Programm »Ein-Byte-Integer« (Listing 2) verwendet diese Routine zur Konvertierung der Zahl $80 (dezimal: 128; binär: 10000000). Das Y-Register wird mit $80 geladen und die Routine aufgerufen. Zur Überprüfung der Konvertierung wird der Inhalt des FAC anschließend mit den Routinen FACSTR und STROUT in einen ASCII-String gewandelt und auf dem Bildschirm ausgegeben.

Ein-Byte-Werte einlesen ohne Vorzeichen

Eingabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= Wert

```
ldy #(Wert)     ;
jsr EINPOS      ; $b3a2
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Wert

Ein-Byte-Integer (vorzeichenbehaftet) nach Fließkomma ($BC3C)

Wenn Sie das Programm »Ein-Byte-Integer« (Listing 2) mit Ihrem Assembler eingeben und mit »SYS 49152« starten, stellen Sie fest, daß auf dem Bildschirm zwei Zahlen unmittelbar hintereinander ausgegeben werden: 128 und -128. Für die Ausgabe der Zahl -128 ist der zweite Programmteil verantwortlich, der die Routine EINNEG ($BC3C) zur Wandlung des Integerwertes $80 verwendet. Diese Routine konvertiert einen vorzeichenbehafteten Ein-Byte-Integer-Wert in eine Fließkommazahl.

Was unter vorzeichenbehaftet zu verstehen ist, wurde bereits kurz angedeutet: Das oberste Bit der Zahl wird als Vorzeichen verwendet. Ist es gesetzt, handelt es sich um eine negative, ansonsten um eine positive Zahl. Da dieses Bit bei der Zahl $80 gesetzt ist, interpretiert die Routine EINNEG $80 als die negative Zahl -128. Zur Verdeutlichung: Vorzeichenbehaftet entspricht $81 dem Wert -127, $82 dem Wert -126 und so weiter.

Beachten Sie bitte, daß vor Benutzung von EINNEG der Akku mit dem jeweiligen Wert geladen werden muß, im Gegensatz zur Routine EINPOS, der die Zahl im Y-Register übergeben wird.

Ein-Byte-Werte einlesen mit Vorzeichen

Eingabe:

Akku	= Wert
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda #(Wert)     ;
jsr EINNEG      ; $bc3c
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Wert

Zwei-Byte-Integer (positiv) nach Fließkomma ($BC49)

In vielen Assemblerprogrammen sind Ein-Byte-Werte unzureichend zur Darstellung der benötigten Zahlen. Ein Programm, das zum Beispiel mit hochauflösender Grafik arbeitet, muß mit Zwei-Byte-Integerzahlen arbeiten, um beliebige Bildschirmpositionen erfassen zu können, da die Grafikauflösung in der Y-Richtung 320 Punkte beträgt und der Wert 255 - die Obergrenze der Ein-Byte-Darstellung - daher überschritten werden kann.

Mit Hilfe der Routine ZWEPOS können positive Zwei-Byte-Integerzahlen (0...65535) in die entsprechende Fließkommazahl konvertiert werden. Vor dem Aufruf wird die jeweilige Zahl nach $62/$63 übertragen (Achtung: $62=High und $63=Low, also zuerst High- und dann Low-Byte!), das X-Register direkt mit $90 geladen und das Carry-Flag gesetzt. Das Programm »Zwei-Byte-Integer« (Listing 3) zeigt die Anwendung dieser Routine zur Konvertierung und Ausgabe (mit FACSTR und STROUT) der Zahl $FFFF.

Zwei Byte einlesen ohne Vorzeichen

Eingabe:

Akku	= unbenutzt
X-Register	= $90
Y-Register	= unbenutzt
Carry	= gesetzt

```
lda #<(Wert)    ; Low-Byte-Wert
sta $63         ; FAC 2.Mantisse
lda #>(Wert)    ; High-Byte-Wert
sta $62         ; FAC 1.Mantisse
sec             ; Carry set
jsr ZWEPOS      ; $bc49
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Wert

Zwei-Byte-Integer (vorzeichenbehaftet) nach Fließkomma ($BC44)

Nach dem Aufruf mit »SYS 49152« gibt dieses Programm ebenso wie das Programm »Ein-Byte-Integer« zwei Zahlen unmittelbar hintereinander aus, und zwar die Zahlen 65535 und -1. Der erste Programmteil benutzt die besprochene Routine ZWEPOS zur Ausgabe von $FFFF ($FFFF=65535), der zweite Teil die Routine ZWENEG ($BC44). ZWENEG faßt die übergebene Zahl als vorzeichenbehafteten Wert auf. In dieser Darstellungsart entspricht $FFFF der größten mit zwei Byte darstellbaren negativen ganzen Zahl (nicht der größten darstellbaren positiven, da das gesetzte oberste Bit eine negative Zahl anzeigt), das heißt der Zahl -1.

Die Übergabeparameter dieser Routine sind mit ZWEPOS identisch, abgesehen davon, daß das Carry-Flag diesmal nicht gesetzt, sondern gelöscht werden muß. Die Zahl wird ebenfalls in $62/$63 (High/Low) übergeben, das X-Register mit $90 geladen und das Carry-Flag gelöscht. Wie Sie vielleicht bemerkt haben, ensprechen die Speicherzellen, in denen der Zwei-Byte-Wert übergeben wird, den ersten beiden Bytes der Mantisse des FAC. Beim C16 beziehungsweise beim C128 wird der gleiche Teil des FAC zur Übergabe benutzt. Die entsprechenden Speicherstellen sind daher

ebenfalls $62/$63 beim C 16 und $64/$65 beim C 128. Die entsprechenden Einsprungadressen finden Sie in der Tabelle am Ende dieses Artikels.

Zwei Byte einlesen mit Vorzeichen

Eingabe:

Akku	= unbenutzt
X-Register	= $90
Y-Register	= unbenutzt
Carry	= gelöscht

```
lda #<(Wert)     ; Low-Byte-Wert
sta $63          ; FAC 2.Mantisse
lda #>(Wert)     ; High-Byte-Wert
sta $62          ; FAC 1.Mantisse
clc              ; Carry clear
jsr ZWENEG       ; $bc44
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Wert

Fließkomma nach Integer ($BC9B)

In allen drei Computern befinden sich zusätzlich Routinen zur Umwandlung von Drei- und Vier-Byte-Integerzahlen ins Fließkommaformat. Da diese jedoch nur selten in Assemblerprogrammen verwendet werden, verzichte ich auf eine ausführliche Darstellung dieser Routinen.

Wir benötigen jedoch unbedingt eine Routine, die den umgekehrten Weg geht, das heißt, die eine Fließkommazahl ins Integerformat wandelt, da es nach verschiedenen Rechenoperationen mit Fließkommazahlen oft einfacher ist, mit dem Ergebnis in Integerform weiterzuarbeiten, als die in Assembler schwer zu verarbeitende Fließkommaform zu verwenden.

Die Routine FLIINT wandelt jede beliebige Fließkommazahl ins Integerformat um. Beachten Sie jedoch bitte, daß die Wandlung nur bei Fließkommazahlen kleiner als 231 (= 2.14 * 109) fehlerfrei funktioniert. Der Grund dafür ist die begrenzte Byteanzahl, die zur Aufnahme des Integerergebnisses verwendet wird (vier Byte).

Da mit Integerzahlen nur ganze Zahlen dargestellt werden können, werden Nachkommastellen bei der Wandlung abgeschnitten (aus 5249,57 wird daher 5249).

FAC in Integer wandeln

Eingabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
jsr FLIINT       ; $bc9b
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Inhalt zerstört

Das Programm »Fließkomma« (Listing 4) wandelt eine Fließkommazahl in einen Integerwert und gibt ihn auf dem Bildschirm aus. Als Fließkommazahl wird PI verwendet und mit der Routine KONFAC in den FAC übertragen. Der Aufruf von FLIINT wandelt diese Fließkommazahl ins Integerformat. FLIINT benötigt außer der zu konvertierenden Fließkommazahl im FAC keine weiteren Übergabeparameter.

Das Integerergebnis befindet sich ebenfalls im FAC, in den vier Byte der Mantisse. In $65 wird das niederwertigste, in $62 das höchstwertige Byte des Ergebnisses abgelegt. Ergibt die Wandlung zum Beispiel die Zahl $20FA, befindet sich in $65 der Wert $FA und in $64 der Wert $20.

Da die Integerwandlung von PI das ganzzahlige Ergebnis drei liefert, interessiert uns nur das niederwertigste Byte $65. Zur Ausgabe des Ergebnisses wird das Y-Register mit diesem Byte geladen, die Routine EINPOS zur Wandlung nach Fließkomma verwendet und diese mit FACSTR und STROUT ausgegeben.

Dieser Weg ist zweifellos sehr umständlich. Wenn Sie sich unmittelbar davon überzeugen wollen, daß die Routine FLIINT die Zahl PI in die Integerzahl drei konvertiert und dabei die Nachkommastellen abschneidet, so fügen Sie im Programm »Fließkomma« nach dem Befehl »JSR FLIINT« bitte ein »BRK« ein. Nach dem Start mit »SYS 49152« verzweigt das Programm nach der Wandlung sofort in den Monitor (den Sie zuvor natürlich laden und aktivieren müssen!). Schauen Sie sich mit dem Monitor anschließend $62 bis $65 an, die Mantisse des FAC. Sie werden feststellen, daß $65 den Wert drei (=$03) enthält und die höherwertigen Ergebnisbytes den Wert 0.

Direkte Bildschirmausgabe von positiven Integerzahlen ($BDCD)

Die Routine INTOUT erlaubt die direkte Ausgabe von positiven Zwei-Byte-Integerzahlen auf dem Bildschirm. Die Integerzahl wird in den Registern übergeben (X=Low-Byte; Akku=High-Byte). Es handelt sich dabei keineswegs um eine neue Routine, da die dieser Adresse folgenden Befehle nur die bekannten Routinen ZWEPOS, FACSTR und STROUT nacheinander aufrufen.

Das Programm »Integerausgabe« (Listing 5) demonstriert den Einsatz dieser Routine. Das X-Register wird mit dem Low-Byte und der Akku mit dem High-Byte von $F02A geladen. Der Aufruf von INTOUT führt unmittelbar zur Ausgabe der Zahl 61482.

Diese Routine kann in den verschiedensten Assemblerprogrammen vielseitig verwendet werden (Textverarbeitung: Ausgabe von Seite, Zeile und Spalte; Spiele: Ausgabe der Punktzahl, des Levels und so weiter).

Ausgabe von Zwei-Byte-Werten

Eingabe:

FAC	= Integerzahl
Akku	= High-Byte-Wert
X-Register	= Low-Byte-Wert
Y-Register	= unbenutzt

```
lda (Wert)       ; FAC 4.Mantisse $65
ldx (Wert)       ; FAC 3.Mantisse $64
jsr INTOUT       ; $bdcd
```

Ausgabe:

Akku	= unbenutzt
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Inhalt zerstört

Beispiel:
Übergabe der Zahl in das Register

```
lda #$1
ldx #$20
jsr INTOUT
```

Ergebnis:
288 auf dem Bildschirm

Grundrechenroutinen

Nachdem wir nun alle häufig benötigten Konvertierungs- und Ausgaberoutinen kennen, können wir uns den eigentlichen Arithmetikroutinen zuwenden, und zwar zuerst den vier Grundrechenarten.

Die vier vorgestellten Routinen arbeiten prinzipiell gleich: Zwei Fließkommazahlen werden durch einen der Operatoren »+-*/« miteinander verknüpft. Die beiden Zahlen müssen sich vor dem Aufruf der benötigten Routine im FAC beziehungsweise ARG befinden. Das Ergebnis der Operation befindet sich nach dem Aufruf immer im FAC (im Fließkommaformat).

Sehr wichtig ist folgende Bedingung: Vor dem Aufruf einer der Routinen muß sich im Akku der Exponent des FAC befinden ($61). Wir werden Routinen kennenlernen, die für den Transport von Konstanten in den FAC oder das Kopieren des ARG-Inhalts in den FAC zuständig sind. Nach der Rückkehr aus diesen Routinen befindet sich der FAC-Exponent immer im Akku. Achten Sie bitte darauf, daß dieser Akkuinhalt bis zum Aufruf der gewünschten Arithmetikroutine auch erhalten bleibt und nicht durch LDA..., TXA etc. zerstört wird.

Das Programm »Grundrechnen« (Listing 6) demonstriert die Anwendung der vier Grundrechenarten. In diesem Programm wird die noch nicht erwähnte Routine FACARG ($BC0C) verwendet, die eine im FAC enthaltene Fließkommazahl nach ARG kopiert. Die entsprechende Adresse dieser Routine für den C 16 finden Sie am Ende des Artikels. Sollten Sie einen C 128 besitzen, können Sie diese Routine entweder selbst im ROM suchen, oder aber eine Schleife verwenden und den Inhalt des FAC »per Hand« nach ARG kopieren. Sie verzichten dann jedoch auf eine exakte Rundung der Zahl, die die Routine FACARG durchführt, bevor die gerundete Zahl nach ARG kopiert wird.

FAC ins ARG kopieren

```
Eingabe:
        FAC         = Wert
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

jsr FACARG      ; $bc0c
                ; bzw ARGFAC $bbfc

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Wert
        ARG         = Wert
```

Der Programmablauf: Gehen wir davon aus, daß Sie die Fläche Ihres Grundbesitzes errechnen wollen. An einer Lagerhalle mit einer Fläche von 2000 m² sind Sie zu einem Drittel beteiligt. Um den Anteil zu ermitteln, müssen wir die Divisionsroutine DIV ($BB12) verwenden.

DIV teilt ARG durch FAC und legt das Ergebnis in FAC ab. Vor Aufruf von DIV muß daher 2000 im ARG und drei im ARG abgelegt werden. Im Programmteil »Anteil berechnen« wird 2000 in $62/$63 abgelegt, das X-Register mit der Zahl $90 geladen und ZWEPOS aufgerufen. Wie wir wissen, befindet sich nach dieser Vorbereitung die Zahl 2000 im Fließkommaformat im FAC.

Diese Fließkommazahl wird durch den Aufruf von FACARG nach ARG kopiert, bevor wir die Zahl drei, mit Hilfe der Routine EINPOS ins Fließkommaformat gewandelt, im FAC ablegen. Der folgende Aufruf von DIV teilt ARG (2000) durch FAC (3) und legt das Ergebnis (666,666667) im FAC ab.

ARG durch FAC = FAC

```
Eingabe:
        FAC         = Teiler
        ARG         = Wert
        Akku        = Exponent d.FAC
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

lda $61         ; Exponent laden,
                  wenn nicht schon im Akku vorhanden
jsr DIV         ; $bb12

Ausgabe:
        Akku        = Eponent d.FAC
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Ergebnis
        ARG         = Wert
```

Zur Kontrolle wollen wir dieses Ergebnis auf dem Bildschirm mit FACSTR und STROUT ausgeben. Da das Ergebnis in den folgenden Rechnungen gebraucht wird, FACSTR den Inhalt des FAC jedoch zerstört, rufen wir zuvor ein weiteres Mal FACARG auf, um den FAC-Inhalt nach ARG zu kopieren. Nachdem das Ergebnis ausgegeben wurde, wird der Akku mit 13 geladen und BSOUT aufgerufen. BSOUT ist eine Betriebssystemroutine, die das im Akku übergebene Zeichen auf dem Bildschirm ausgibt. Die Einsprungadresse $FFD2 ist dank der erwähnten Sprungtabelle der Betriebssystemroutinen für alle drei Computer identisch.

13 ist der Code für »Carriage Return«. Die Ausgabe dieses Zeichens bewirkt einen Zeilenvorschub und sorgt damit für die optische Trennung der verschiedenen Ergebnisse, die »Grundrechnen« berechnet und ausgibt.

Eine weitere Besonderheit des Ausgabeunterprogramms ist der Aufruf von zwei weiteren Unterprogrammen, die FAC nach PUFFER kopieren beziehungsweise PUFFER nach beendeter Ausgabe wieder nach FAC kopieren. Der Grund ist die erwähnte Zerstörung des Inhaltes von FAC bei Aufruf von FACSTR.

Der nächste Programmteil verwendet die Additionsroutine ADD ($B86A), die FAC und ARG addiert, wobei das Ergebnis wiederum in FAC abgelegt wird. Gehen wir davon aus, daß Ihr Grundbesitz außer einem Drittelanteil an HAUS1 aus einem weiteren Haus mit einer Fläche von 5286 m² besteht, das Ihnen allein gehört. Diese Fläche soll zum vorigen Ergebnis addiert werden.

ARG plus FAC = FAC

```
Eingabe:
        FAC         = Wert 1
        ARG         = Wert 2
        Akku        = Exponent d. FAC
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

lda $61         ; Exponent laden, wenn nicht schon
                  im Akku vorhanden
jsr ADD         ; $b86a

        Ausgabe:
        Akku        = Eponent d. FAC
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Ergebnis
        ARG         = Wert
```

Zuerst wird der Inhalt von FAC mit FACARG nach ARG kopiert. Anschließend wird mit ZWEPOS die Zwei-Byte-Integerzahl 5286 im Fließkommaformat in FAC abgelegt. Der folgende Aufruf von ADD addiert FAC und ARG. Das Ergebnis wird mit der Ausgaberoutine auf dem Bildschirm ausgegeben (5952,66667), so daß Sie es problemlos mit einem Taschenrechner überprüfen können.

Etwas spät fällt Ihnen nun ein, daß die Grundfläche des zweiten Hauses vor kurzem neu berechnet wurde und das Ergebnis um 52 m^2 niedriger ausfiel als Ihr alter Wert von 5286 m^2. Das bisherige Ergebnis muß daher korrigiert werden, Sie müssen 52 subtrahieren.

Mit den Arithmetikroutinen stellt diese Korrektur kein Problem dar. Wir verwenden die Routine SUB ($B853), die FAC von ARG subtrahiert.

Im dritten Programmteil wird zuerst das bisherige Ergebnis mit FACARG wieder in ARG kopiert. Der Routine EINPOS wird die Zahl 52 übergeben, die dadurch als Fließkommazahl in FAC abgelegt wird. EINPOS kann verwendet werden, da es sich bei 52 um einen Ein-Byte-Wert handelt. Wenn Sie in eigenen Programmen nicht von vornherein wissen, wie groß die zu wandelnden Integerzahlen sind, verwenden Sie am besten immer die Routine ZWEPOS.

Nach der Wandlung wird die Routine SUB aufgerufen, die FAC=ARG-FAC rechnet, in unserem Fall 5952,66667-52. Das Ergebnis 5900.66667 wird anschließend ausgegeben.

ARG minus FAC = FAC

Eingabe:

FAC	= Subtrahent
ARG	= Wert
Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr SUB          ; $b853
```

Ausgabe:

Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Ergebnis
ARG	= Wert

Sie kennen nun die Gesamtfläche der Ihnen gehörenden Grundstücke und wollen deren Wert berechnen. Sie gehen dabei von einem durchschnittlichen Quadratmeterpreis von 228 Mark aus. Die Fläche von 5900,66667 m^2 muß daher mit 228 multipliziert werden.

Die Fläche, die sich noch in FAC befindet, wird mit FACARG nach ARG kopiert, bevor die Ein-Byte-Integerzahl 228 an EINPOS übergeben wird. Durch Aufruf der Routine MULT ($BA2B) werden FAC und ARG miteinander multipliziert. Als Ausgabe erhalten Sie das stolze Ergebnis von 1345352 Mark.

Sie sehen, die Benutzung der Grundrechenroutinen in Assembler stellt keine besonderen Probleme, wenn beachtet wird, daß die Ausgabe von Zwischenergebnissen mit FACSTR und STROUT den Inhalt von FAC zerstört und dieser daher vor der Ausgabe gerettet werden muß.

EXP ($BF7B)

EXP wurde noch nicht erwähnt. Diese Routine arbeitet ebenfalls mit zwei Fließkommazahlen in FAC und ARG. EXP potenziert den ARG mit dem im FAC enthaltenen Exponenten. (FAC=3 und ARG=2 ergibt FAC=8).

ARG mal FAC = FAC

Eingabe:

FAC	= Multiplikator
ARG	= Wert
Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr MULT         ; $ba2b
```

Ausgabe:

Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Ergebnis
ARG	= Wert

ARG hoch FAC = FAC

Eingabe:

FAC	= Exponent
ARG	= Wert
Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr EXP          ; $bf7b
```

Ausgabe:

Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Ergebnis
ARG	= Wert

Die Funktionen

Zur komfortablen Benutzung der Arithmetikfunktionen benötigen wir außer den Grundrechenarten die eingebauten Fließkommafunktionen. Das Programm »Funktionen« (Listing 7) demonstriert den Gebrauch der Funktionen SIN(X), COS(X), SQR(X), LOG(X) und EXP(X). Vor dem Aufruf einer dieser Funktionen muß sich das Argument X der Funktion im FAC befinden. Das Ergebnis des Aufrufs befindet sich anschließend wieder im FAC.

Das Programm »Funktionen« verwendet als Argument die Zahl zehn. Diese Zahl wird der Routine EINPOS im Y-Register übergeben. Nach dem Aufruf von EINPOS befindet sich die Fließkommazahl zehn im FAC und die gewünschte Funktion wird aufgerufen. Das Ergebnis des Funktionsaufrufs wird wie gewohnt mit FACSTR und STROUT auf dem Bildschirm ausgegeben.

```
COS(10) =       -0,544021111 (Adresse: $E264)
SIN(10) =       -0,839071529 (Adresse: $E26B)
SQR(10) =        3,16227766  (Adresse: $BF71)
LOG(10) =        2,30258509  (Adresse: $B9EA)
EXP(10) =  22026,4658        (Adresse: $BFED)
```

Wenn Sie sich davon überzeugen wollen, daß die ausgegebenen Ergebnisse korrekt sind, so können Sie im Direktmodus eingeben:

```
PRINT SIN(10):PRINT COS(10):PRINT SQR(10):
PRINT LOG(10):PRINT EXP(10)
```

und die Ergebnisse mit den Ausgaben des Assemblerprogramms vergleichen.

Funktionen
Eingabe:

FAC	= Wert
Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr (Funktion)   ;
```

Ausgabe:

Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Ergebnis

Wie Sie sehen, ist der Aufruf einer Funktion noch einfacher als die Anwendung einer Grundrechenroutine. Es genügt, das Funktionsargument vor dem Aufruf in den FAC zu transportieren. Außer den im Programm »Funktionen« verwendeten existiert noch eine Vielzahl weiterer Funktionen, die auf die gleiche Weise auf ein beliebiges Argument X angewendet werden können (TAN, RND etc.).

Sollten Sie mit der Wirkungsweise einiger Funktionen nicht vertraut sein, empfehle ich Ihnen das Studium des Handbuchs, in dem diese bei der Basic-Programmierung beschrieben werden. Denken Sie daran, daß diese Funktionen im Basic-Interpreter integriert sind, daß wir von Assembler aus keine neuen, sondern nur die auch von Basic aus nutzbaren Funktionen ansprechen.

Schnittstellen zu Basic

Bisher wurde die Anwendung der Rechenroutinen und Funktionen in reinen Assemblerprogrammen demonstriert. Eine Hauptanwendung der Maschinenprogrammierung ist jedoch oftmals der Ersatz zeitkritischer Basic-Programmteile durch Assemblerroutinen. Bisher wissen wir jedoch noch nicht, wie von Basic aus Parameter an ein Maschinenprogramm übergeben werden können.



Eine Methode zur Übergabe dürfte jedem von Ihnen bekannt sein: Der Befehl POKE. Mit POKE können wir beliebige Integerzahlen übergeben. Am einfachsten ist die Übergabe einer Ein-Byte-Integerzahl, zum Beispiel durch POKE 250,10. Um mit diesem Befehl einen Zwei-Byte-Wert zu übergeben, kann dieser in ein Low- und ein High-Byte aufgesplittet werden:

```
100 rem *unterprogramm zur parameteruebergabe*
110 rem parameter wird in 'wert' uebergeben
120 hb=int(wert/256) : rem high-byte
130 lb=wert-hb*256 : rem low-byte
140 poke 250,lb:poke 251,hb : rem uebergabe
150 return
```

Dieses Unterprogramm kann vom Hauptprogramm jederzeit mit »GOSUB 120« aufgerufen werden, wenn der zu übergebende Parameter zuvor der Variablen »wert« zugewiesen wurde.

Beispiel:

```
10 wert=1000 : gosub 120
```

Dieser Aufruf teilt die Zahl 1000 in das High-Byte drei und das Low-Byte 232. Das Low-Byte wird in Speicherstelle 250 und das High-Byte in 251 übergeben. Der Maschinenroutine wird der Zwei-Byte-Wert in der gewohnten Form Low-Byte/High-Byte übergeben.

Das Schlüsselloch

Diese Routine sollten Sie jedoch schnellstens wieder vergessen. Die vorgestellte Methode hat gleich mehrere Nachteile. Zum einen ist es schlichtweg unsinnig, zur Übergabe eines Parameters an eine Maschinenroutine, die einen zeitkritischen Programmteil beschleunigen soll, ein aufwendiges Basic-Unterprogramm zu verwenden. Die Rechnungen und der Unterprogrammaufruf benötigen soviel Zeit, daß der Zeitvorteil der Maschinenroutine wahrscheinlich zumindest teilweise dadurch wieder zunichte gemacht wird, wie im folgenden Beispiel:

```
10 for i=1 to 1000
20   wert=i      : wertzuweisung
30   gosub 120   : uebergabe des wertes
40   sys 49152   : aufruf der maschinenroutine
40 next
```

In diesem Programm werden einer Maschinenroutine nacheinander die Werte eins bis 1000 in einer Schleife übergeben. Das heißt, daß der Basic-Interpreter 1000mal das komplette Basic-Unterprogramm durchlaufen muß!

Der zweite Nachteil unserer Übergaberoutine ist noch schwerwiegender. Mit der vorgestellten Routine ist es völlig unmöglich, Fließkommazahlen an ein Maschinenprogramm zu übergeben. (Außer, Sie haben vor, eine Basic-Übergaberoutine zu schreiben, die eine beliebige Zahl ins Fließkommaformat wandelt und mit POKEs übergibt. Vom Zeitaufwand her gesehen ist es dann jedoch wirklich angebrachter, die Berechnungen in »reinem« Basic durchzuführen. Selbst aufwendige Rechnungen können nur schwerlich langsamer als ein solches Konvertierungsprogramm sein.)

Die USR-Funktion

Basic besitzt eine Schnittstelle zu Maschinenprogrammen, die hervorragend zur Zahlenübergabe, insbesondere zur Übergabe reeller Zahlen (also nicht nur Integerzahlen), geeignet ist, die Funktion USR(X). Dieser Befehl, der im Handbuch des C 64 nur unzureichend erläutert ist, legt eine beliebige Zahl X im Fließkommaformat im FAC ab.

Vor der Verwendung der Funktion USR muß der sogenannte USR-Vektor »verbogen« werden. Dieser Vektor besteht aus zwei Speicherstellen (785/786 beim C 64), in die in der bekannten Form Low-Byte/High-Byte die Startadresse der Maschinenroutine gePOKEt wird.

Die Anwendung der Funktion USR demonstriert das Programm »USR-Vektor« (Listing 8). Das Programm besteht aus zwei Teilen. Teil 1 beginnt ab Adresse $C000 (dezimal 49152), Teil 2 ab Adresse $C100 (dezimal 49408). Der erste Programmteil hat die Aufgabe, die Wurzel einer beliebigen mit USR übergebenen Zahl zu berechnen.

Vor dem Aufruf dieses Programmteils muß der USR-Vektor auf die Startadresse $C000 gerichtet werden. In Speicherstelle 785 wird das Low- und in 786 das High-Byte der Startadresse gepokt. Low- und High-Byte des dezimalen Wertes 49152 können in Basic mit Hilfe der beschriebenen Routine (Zeile 120-130) ermittelt werden. In unserem Fall ergibt sich als Low-Byte der Wert null und als High-Byte die 192. Der USR-Vektor wird daher mit folgenden Befehlen auf den ersten Programmteil gerichtet:

```
POKE 785,0:POKE 786,192
```

Geben Sie diese Befehle nach dem Assemblieren des Programms bitte im Direktmodus ein. Um die Wurzel von 20 zu berechnen, geben Sie anschließend ebenfalls im Direktmodus ein:

```
PRINT USR(20)
```

Auf dem Bildschirm wird als Ergebnis die Zahl 4,47213595 ausgegeben. Nach dem Aufruf der Funktion USR wurde der übergebene Parameter 20 ins Fließkommaformat gewandelt im FAC abgelegt. Anschließend erfolgte ein indirekter Sprung über den USR-Vektor (JMP(785)), der auf unseren ersten Programmteil zeigt. Dieser indirekte Sprung kann mit einem »SYS 49152« verglichen werden, das heißt unsere Routine wurde gestartet.

Das Ei des Kolumbus

Die Routine ruft die SQR-Funktion auf. Das Argument X ist bereits im FAC vorhanden. SQR berechnet die Wurzel von X und legt sie ebenfalls im Fließkommaformat wieder im FAC ab. Da das Ergebnis im zweiten Programmteil benötigt wird, kopiert die Routine FACPUF den Inhalt des FAC in den Puffer am Programmende, bevor die Rückkehr nach Basic mit RTS erfolgt. Der Basic-Befehl PRINT führt nun zur Ausgabe der im FAC enthaltenen Fließkommazahl, das heißt der Wurzel von 20.

USR übergibt somit eine Zahl nach Wandlung ins Fließkommaformat im FAC an ein Maschinenprogramm, das anschließend über den USR-Vektor aufgerufen wird und das Ergebnis von Rechenoperationen ebenfalls im FAC an Basic zurückübergibt.

Das im FAC übergebene Ergebnis kann von Basic beliebig weiterverarbeitet, zum Beispiel einer Variablen zugewiesen werden:

```
A=USR(1000):PRINT A
```

Die Eingabe dieser Befehle im Direktmodus führt zur Ausgabe von 31,6227766. Das Ergebnis der Rechenoperationen wurde der Variablen A zugewiesen.

Bevor wir uns dem zweiten Programmteil zuwenden, geben Sie bitte noch ein letztes Mal ein:

```
PRINT USR(20)
```

damit unser erster Programmteil die Wurzel von 20 in den Puffer kopiert. Der zweite Programmteil soll eine ebenfalls mit USR zu übergebende Zahl mit dieser Wurzel multiplizieren.

Da sich Teil 2 ab Adresse $C100 im Speicher befindet, muß der USR-Vektor auf diese neue Startadresse gerichtet werden:

```
POKE 785,0:POKE 786,193
```

Geben Sie ebenfalls im Direktmodus ein:

```
PRINT USR(5.3)
```

Als Ausgabe erhalten wir 23,7023206, was exakt der Multiplikation von 5,3 mit der zuvor errechneten Wurzel von 20 entspricht. Dem zweiten Programmteil wurde nach dem Aufruf mit USR die Zahl 5,3 im FAC übergeben. Diese Zahl wird durch FACARG nach ARG kopiert, bevor der Inhalt des Puffers, die von Teil 1 berechnete Wurzel von 20, nach FAC kopiert wird. Der Aufruf von MULT führt zur Multiplikation von FAC mit ARG und Ablage des Ergebnisses im FAC. Dieses Ergebnis wird mit dem PRINT-Befehl ausgegeben.

USR mit mehreren Parametern

Nachdem die prinzipielle Funktionsweise des Befehls USR nun deutlich wurde, stelle ich eine von vielen denkbaren Methoden vor, beliebig viele Parameter mit USR an ein Maschinenprogramm zu übergeben.

Das Programm »USR-Vektor.2« (Listing 9) verwendet folgende Methode: Das aufrufende Basic-Programm übergibt mit POKE in Speicherzelle 250 die Anzahl der Übergabeparameter. Das Maschinenprogramm kopiert nach jedem Aufruf mit USR die im FAC übergebene Fließkommazahl in den Puffer am Programmende. Um ein Überschreiben zu vermeiden, muß jede weitere Fließkommazahl fünf Byte hinter der zuletzt kopierten in den Puffer geschrieben werden.

In diesem Fall ist die Programmierung einfacher als die eines sich selbst verändernden Programms.

Jedesmal, wenn eine Fließkommazahl in den Puffer kopiert wurde, wird der Befehl STA PUFFER,X verändert. Die Adresse PUFFER wird um den Wert fünf inkrementiert, so daß sie unmittelbar hinter die zuletzt abgespeicherten Daten zeigt. Bei jeder Parameterübernahme wird die Speicherzelle 250 dekrementiert. Der Wert null zeigt an, daß alle zu übergebenden Parameter in den Puffer kopiert wurden und der Programmteil »Rechnen« wird angesprungen, der die Aufgabe besitzt, alle übernommenen Zahlen zu addieren.

»Rechnen« verwendet ebenfalls diese Programmiertechnik. Der Inhalt von FAC (die letzte übergebene Zahl) wird mit FACARG nach ARG kopiert, der erste übergebene Parameter aus dem Puffer nach FAC kopiert und ADD angesprungen. Anschließend wird die Adresse im Befehl LDA PUFFER,X um fünf inkrementiert, so daß sie auf die nächste Fließkommazahl im Puffer weist.

Bevor diese zur bisherigen Summe addiert wird, muß überprüft werden, ob die Adresse im Befehl LDA PUFFER,X bereits soweit erhöht wurde, daß sie mit der Adresse im Befehl STA PUFFER,X identisch ist. Wenn ja, wurde die letzte übergebene Fließkommazahl bereits behandelt und die Aufsummierung wird durch die Rückkehr nach Basic beendet. Im FAC befindet sich nun die Summe aller übergebenen Zahlen.

Wenn Sie dieses Programm assembliert haben, geben Sie bitte folgendes Programm ein:

```
100 poke 785,0:poke 786,192
:rem usr-vektor auf routine
110 an=100:rem anzahl aufzusummierender elemente
120 poke 250,an:rem anzahl in 250 uebergeben
130 for i=1 to an:s=usr(i):next:rem parameter
140 print s:rem ergebnis der summierung
```

Erklärungsbedürftig in diesem Programm ist die Zeile 130. In der Schleife werden der Maschinenroutine der Reihe nach die Zahlen eins bis 100 übergeben. Nach jedem Aufruf mit USR(I) wird der gerade übergebene Wert der Variablen S zugewiesen. Nach der letzten Übergabe und der darauf folgenden Ausführung des Programmteils »Rechnen« befindet sich im FAC die Summe, die nun mit S=USR(I) der Variablen S zugewiesen wird. Der Befehl PRINT S führt zur Ausgabe der Summe 5050, die mit den Basic-Befehlen:

```
for i=1 to 100:x=x+i:next:print x
```

überprüft werden kann. Ich hoffe, »Puritaner« mögen mir die verwendete Programmiertechnik verzeihen. Dieses Programm kann selbstverständlich auch unter Verwendung der indirekt indizierten Adressierung geschrieben werden »LDA

(POINTER),Y«. Was demonstriert werden sollte: daß mit dem USR-Befehl prinzipiell die Übergabe beliebig vieler reeller Zahlen an eine Maschinenroutine möglich ist (wobei die maximale Parameteranzahl bei vorliegendem Programm etwa 800 beträgt, da ansonsten der Puffer durch Überschreitung der Adresse $CFFF »überläuft«).

Welche Rechenoperationen Sie mit den übergebenen Parametern ausführen, ob Sie sie aufsummieren, multiplizieren oder statistische Berechnungen wie zum Beispiel die Berechnung von Mittelwert oder Streuung durchführen, bleibt völlig Ihnen überlassen.

CHKKOM, GETBYT, FRMNUM, ADRFOR, GETADR

Wie wir sahen, können mit USR problemlos »krumme« Werte an eine Maschinenroutine übergeben werden. Wir sahen ebenfalls, daß die Übergabe mehrerer Parameter erheblich aufwendiger ist. Der Basic-Interpreter stellt uns jedoch noch weitere Routinen zur Verfügung, mit denen eine unkomplizierte Übergabe beliebig vieler Parameter möglich ist.

Es sind Routinen, die keine »offizielle« Schnittstelle wie die USR-Funktion darstellen, sondern die der Basic-Interpreter selbst bei der Abarbeitung eines Programms benötigt. Mit diesen Routinen ist es möglich, Parameter direkt aus dem Basic-Text zu lesen, die hinter einem SYS-Befehl aufgereiht übergeben werden, zum Beispiel mit SYS 49152,10,20,30.

Zuerst müssen wir wissen, daß der Basic-Interpreter in den Speicherstellen 122/123 (C 64) einen Pointer auf den Basic-Text verwaltet, der auf das aktuelle Zeichen weist. Nach Abarbeitung des SYS-Befehls, mit dem unsere Routine aufgerufen wird, weist dieser Pointer auf das der Adresse (49152) folgende Komma. Da hinter dem Komma der erste Übergabeparameter folgt, benötigen wir eine Routine, die es ermöglicht, das Komma einzulesen und den Textpointer auf den ersten Parameter zu setzen.

Zum Einlesen eines Kommas wird die Routine CHKKOM ($AEFD) verwendet. Durch Aufruf von CHKKOM wird das aktuelle Zeichen eingelesen, auf das der Textpointer zeigt und überprüft, ob es sich bei diesem Zeichen um ein Komma handelt. Fällt der Test negativ aus, wird ein »Syntax error« ausgegeben. War das eingelesene Zeichen tatsächlich ein Komma, wird der Textpointer inkrementiert – und weist somit auf das nächste einzulesende Zeichen –, bevor das Unterprogramm CHKKOM mit RTS beendet wird.



Dank CHKKOM können wir mehrere Parameter durch ein definiertes Zeichen voneinander unterscheiden und überprüfen, ob der Benutzer unserer Routine die verschiedenen Parameter korrekt mit Kommatas getrennt hat. Das Einlesen und die Überprüfung des Trennzeichens nützt uns jedoch wenig, solange wir nicht an die eigentlichen Parameter herankommen.

Die einfachste Routine zum Einlesen eines Parameters nennt sich GETBYT ($B79E). Vor dem Aufruf von GETBYT muß der Textpointer auf das erste Zeichen des Parameters weisen. Da der Aufruf von CHKKOM den Textpointer inkrementiert und dieser damit auf das erste Zeichen hinter (!) dem Komma weist, ist diese Bedingung erfüllt. Eine zweite Bedingung besteht darin, daß der übergebene Parameter ein Ein-Byte-Wert sein muß, entweder eine Ein-Byte-Zahl (null bis 255) oder aber eine Variable, der ein Ein-Byte-Wert zugewiesen wurde (10 a=200:sys 49152,a), da sonst ein »syntax error« ausgegeben wird. Nach dem Einlesen des Parameters weist der Textpointer übrigens ebenso wie bei den noch zu besprechenden Routinen FRMNUM und GETPOS hinter das letzte Zeichen der Zahl beziehungsweise Variablen, bei der Übergabe mehrerer durch Kommatas getrennter Variablen somit auf das nächste Komma.

SYS mit mehreren Parametern

GETBYT übergibt den eingelesenen Parameter im X-Register. Mit CHKKOM und GETBYT können beliebig viele Ein-Byte-Werte übergeben werden, wie das folgende Beispiel zeigt:

```
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;1. parameter in x einlesen
stx PUFFER     ;1. parameter nach puffer
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;2. parameter in x einlesen
stx puffer+1   ;2. parameter nach puffer+1
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;3. parameter in x einlesen
stx puffer+2   ;3. parameter nach puffer+2
```

Diese Routine liest drei einem SYS-Befehl folgende und durch Kommata getrennte Ein-Byte-Werte ein, die in Puffer, Puffer+1 und Puffer+2 abgelegt werden. Die Parameter können sowohl direkt angegebene Zahlen als auch Variablen sein, deren Inhalt GETBYT ermittelt:

1. sys 49152,10,20,30
2. a=10:b=20:c=30:sys 49152,a,b,c
3. b=20:sys 49152,10,b,30

Die Routine FRMNUM ($AD8A) ähnelt der USR-Funktion. FRMNUM liest einen beliebigen numerischen Ausdruck und wertet ihn aus. Das Ergebnis wird im Fließkommaformat im FAC hinterlegt. Außer als Ersatz der USR-Funktion kann FRMNUM in Verbindung mit der Routine ADRFOR ($B7F7) zur einfachen Übergabe von Zwei-Byte-Integerwerten verwendet werden. ADRFOR wandelt die Fließkommazahl im FAC in eine Zwei-Byte-Integerzahl, die im Akku und Y-Register übergeben wird (Y=Low-Byte/Akku=High-Byte).

Mit CHKKOM, GETBYT, FRMNUM und ADRFOR können nach einem SYS-Befehl Ein- oder Zwei-Byte-Werte und auch reelle Zahlen übergeben werden. Das Einlesen eines Zwei-Byte-Wertes erfordert den Aufruf von CHKKOM, FRMNUM und ADRFOR:

```
jsr CHKKOM
jsr FRNUM
jsr ADRFOR
```

Aufruf zum Beispiel mit: sys 49152,1000

Leider nur für den C128 bekannt ist mir die Adresse der Routine GETADR ($880F), die diese drei Routinen nacheinander aufruft. Beim C128 können Sie sich somit zum Einle-

sen einer Zwei-Byte-Zahl in Akku und Y-Register auf den Aufruf von GETADR beschränken.

Eine weitere interessante Routine ist ADRBYT ($B7EB). Diese Routine ruft nacheinander FRMNUM, ADRFOR, CHKKOM und GETBYT auf, liest also zuerst einen Zwei-Byte-Wert, danach das Komma und anschließend einen Ein-Byte-Wert ein, was zum Beispiel für Grafikroutinen interessant sein kann (X-Richtung: 0..319; Y-Richtung: 0..199). Da Sie sich nun sicherlich fragen, ob der ADRFOR folgenden Aufruf von CHKKOM nicht mit den im Akku und Y-Register übergebenen Zwei-Byte-Wert überschreibt: der Wert wird tatsächlich überschrieben. ADRFOR übergibt den Parameter jedoch nicht nur in den beiden Prozessor-Registern, sondern legt ihn zusätzlich noch in den Speicherstellen $14/$15 ab.

Formeln sind erlaubt

Das Programm »Parameter lesen« (Listing 10) demonstriert die Anwendung der beschriebenen Routinen. Geben Sie bitte nach der Assemblierung folgendes Basic-Programm ein:

```
100 SYS 49152,10,2000,5000,60,2*3.4
```

Der Reihe nach werden folgende Zahlen ausgegeben werden:

```
10
2000
60
5000
 6,8
```

Nachdem mit CHKKOM das erste Komma gelesen wurde, wird mit GETBYT die Zahl zehn ins X-Register gelesen. Da der Akku keinen definierten Wert besitzt, wird er mit null geladen, bevor die übergebene Zahl mit INTOUT ausgegeben wird. Erinnern Sie sich daran, daß INTOUT eine Zwei-Byte-Integerzahl ausgibt (X=Low-Byte/Akku=High-Byte).

Das nächste Komma wird eingelesen und danach mit FRMNUM und ADRFOR die Zwei-Byte-Zahl 2000. Da ADRFOR das Low-Byte im Y-Register und das High-Byte im Akku übergibt, die Routine INTOUT das Low-Byte jedoch im X-Register erwartet, wird der Inhalt von Y ins X-Register kopiert.

Mit CHKKOM und ADRBYT werden in einem Zug die Zahl 5000 und die Zahl 60 gelesen, die sich nach ADRBYT im X-Register befindet und mit INTOUT ausgegeben wird (zuvor wird wieder das High-Byte gelöscht, das heißt der Akku mit null geladen).

Auf die zuerst eingelesene Zahl 5000 kann wie erläutert über die Speicherstellen $14/$15 zugegriffen werden. Nachdem sowohl X-Register als auch Akku mit den entsprechenden Inhalten geladen wurden, wird INTOUT aufgerufen.

Der letzte Programmteil zeigt, daß mit FRMNUM beliebige numerische Ausdrücke sehr einfach im Fließkommaformat an den FAC übergeben werden können. Meiner Ansicht nach ist FRMNUM der Funktion USR vorzuziehen, da es nicht in jedem Fall erwünscht ist, daß nach dem Aufruf von USR unweigerlich ein Sprung über den USR-Vektor erfolgt.

Sie besitzen nun alle Mittel, die Sie benötigen, um beliebige Parameter von Basic an eine Maschinenroutine zu übergeben, oder?

Variablen suchen

Nun, eine wesentliche Möglichkeit der Parameterübergabe wurde bisher übergangen: die Übergabe der Strings! Wir können zwar beliebige numerische Werte übergeben, besitzen jedoch vorläufig keine Ahnung, wie ein Maschinenprogramm auch auf Strings zugreifen kann.

Der Zugriff auf Strings ist eigentlich in einem Artikel über Arithmetikroutinen fehl am Platz. Ich erwähne ihn dennoch, weil wir im folgenden die flexibelste Routine zur Parameterübergabe besprechen werden und diese sich sowohl auf numerische Variablen als auch auf Strings bezieht.

GETPOS

GETPOS ($B08B) ist eine der wichtigsten Routinen des Basic-Interpreters. Assemblerunterprogramme für ein Basic-Programm zu schreiben, wird erst mit GETPOS wirklich komfortabel.

Der Routine GETPOS wird ein beliebiger (!) Variablenname übergeben. Diese Routine sucht die zugehörige Variable und übergibt im Akku und Y-Register einen Zeiger auf die Variable. Im Akku befindet sich das Low- und im Y-Register das High-Byte dieses Zeigers.

Um GETPOS verwenden zu können, müssen Sie jedoch wissen, wie der Basic-Interpreter Variablen ablegt:

1. Integer: (Name)(Name)(High)(Low)(.)(.)(.)(Name...
2. Real: (Name)(Name)(5 Byte Fließkomma)(Name...
3. String: (Name)(Name)(Länge)(Adr-Low)
Adr-High)(.)(.)(Name...)

Unabhängig vom Variablentyp werden bei nichtdimensionierten Variablen sieben Byte pro Eintrag in die sogenannte »Variablentabelle« verwendet. Die beiden ersten Bytes sind immer die ersten signifikanten Zeichen des jeweiligen Variablennamens.

Bei Integervariablen folgt der Wert in der ungewohnten Form Low-Byte/High-Byte und die nächsten drei Byte sind ungenutzt. Bei Realvariablen besteht die Zahl selbst aus den Bytes drei bis sieben (Fließkommaformat). Für Strings enthält die Variablentabelle die sogenannten »Stringdescriptoren«, nicht den String selbst. Die Stringdescriptoren bestehen aus einem Byte für die Stringlänge und zwei Byte für die Adresse (im gewohnten Format Low/High), an der sich der String selbst befindet, das heißt einen Zeiger auf den String. Die beiden nächsten Bytes sind ebenso wie bei Integervariablen ungenutzt.

Dimensionierte Felder

Folgende Änderungen müssen bei dimensionierten Variablen beachtet werden: Der Name der Arrayvariablen befindet sich nur einmal am Arrayanfang (zwei Byte). Die folgenden beiden Bytes geben den vom Array belegten Speicherplatz an, ein weiteres Byte die Arraydimension, und Byte sechs und sieben enthalten die Anzahl der Arrayelemente. Nach dieser Beschreibung folgen die einzelnen Elemente des Arrays, wobei jedoch speicherplatzsparender als bei nichtdimensionierten Variablen vorgegangen wird:

1. Integer:
(Var1:Low)(Var1:High)(Var2:Low)(Var2:High)(Var...
2. Real: (Var1: 5 Byte)(Var2: 5 Byte)(Var3: 5 Byte)(Var...
3. String (Var1:Länge)(Var1:Adr-Low)
(Var1:Adr-High)(Var2:Länge..

Wie Sie sehen, sind dimensionierte Arrays weit kompakter aufgebaut als nicht dimensionierte. Die Variablentabelle selbst befindet sich beim C16 und C64 unmittelbar hinter dem Ende des Basic-Programms, beim C128 ab $0400 in Bank 1, der Variablenbank.

Die Strings werden bei allen drei Computern ab dem Ende des RAM-Speichers abwärts angelegt (in Bank 1 beim C128). In diesem »String-Stack« liegen die Strings im ASCII-Format abgelegt unmittelbar hintereinander. Beim C128 existiert zusätzlich zu jedem String ein »Rückwärtszeiger«, der auf die zu diesem String gehörenden Descriptoren weist und die Garbage Collection erheblich beschleunigt:
C16/C64:
MaierMüllerWiedemann...

C128:
Maier(Pointer)Müller(Pointer)Wiedemann(Pointer)...

Der Aufruf von GETPOS liefert immer (!) im Akku und Y-Register einen Pointer (Zeiger) auf das erste Byte des Descriptors der jeweiligen Variablen, unabhängig vom Variablentyp und unabhängig davon, ob es sich um eine dimensionierte oder einfache Variable handelt. Der übergebene Pointer weist daher bei nichtdimensionierten Variablen auf das erste Byte des Variablennamens und bei dimensionierten Variablen auf das erste Byte der Variablen selbst, beziehungsweise bei dimensionierten Stringvariablen auf den Längendescriptor der Stringvariablen.

Wenn GETPOS feststellt, daß noch keine Variable mit dem übergebenen Namen existiert, wird eine neue Variable unter diesem Namen angelegt.

Für die Übergabe aus Basic können die verschiedensten Formate verwendet werden:

1. SYS 49152,I% SYS 49152,I%(2) SYS 49152,I%(J)
2. SYS 49152,R SYS 49152,R(2) SYS 49152,R(J)
3. SYS 49152,S$ SYS 49152,S$(2) SYS 49152,S$(J)

Die zugehörige Assemblerroutine (unabhängig vom Variablentyp!):

```
JSR CHKKOM    ;KOMMA LESEN
JSR GETPOS    ;=> POINTER (AKKU=LOW/Y=HIGH)
```

Mit diesen Kenntnissen können wir nun endlich unsere Arithmetikroutinen effektiv einsetzen. Im Demoprogramm »USR-Vektor.2« wurden Realzahlen aufsummiert, wobei jede Zahl mit einem USR-Aufruf an die Maschinenroutine übergeben wurde. Diese Übergabe (Basic-Schleife!) ist jedoch langsam, daß das Basic-Programm »FOR i=1 to 100:I=I+1: NEXT« fast exakt gleich schnell ist, wie Sie jederzeit ausprobieren können.

Diese langwierige Übergabe entfällt jedoch völlig, wenn die betreffenden Zahlen in einem Array abgelegt werden und Basic dem Assemblerprogramm nur noch den Namen des Arrays mitteilen muß, das die Maschinenroutine nun selbständig bearbeiten kann. Um das Programm komfortabler zu gestalten, sehen wir die Möglichkeit vor, nur einen bestimmten Arrayteil aufzusummieren. Das Basic-Programm muß in diesem Fall zwei Parameter übergeben, die mit GETPOS gelesen werden: den Namen des ersten und den Namen des letzten Arrayelementes, das in die Summierung einbezogen werden soll, also die Intervallgrenzen.



Sollen zum Beispiel die ersten hundert Elemente des Arrays A(..) summiert werden, lautet der Aufruf:

```
SYS 49152,A(1),A(100)
```

Das Programm »Summierung« (Listing 11) demonstriert die Geschwindigkeitsvorteile von Assembler ohne die »Bremse« der Basic-Übergabe. Zweimal hintereinander wird CHKKOM und GETPOS aufgerufen. Im ersten Fall wird durch GETPOS ein Pointer auf das erste zu behandelnde Arrayelement A(1) übergeben, der nach START/START+1 kopiert wird, im zweiten Durchgang wird der auf A(100) - die rechte Begrenzung - übergebene Zeiger nach ENDE/ENDE+1 kopiert.

Das Unterprogramm INFAC verwendet den Zeiger START zum Aufruf von KONFAC, jener Routine, die eine Fließkommakonstante nach ARG kopiert. Der erste Aufruf dieses Unterprogramms kopiert das Element A(1) nach FAC.

Igel und Hase

In der folgenden Hauptschleife wird FAC nach ARG kopiert und der Pointer START(+1) um fünf erhöht. Der Pointer weist anschließend auf das erste Byte der Realzahl A(2). Es folgt ein Vergleich von START(+1) mit ENDE(+1). Wenn START(+1) größer ist als ENDE(+1), wurde die übergebene rechte Arraygrenze bereits überschritten, die Arbeit der Routine ist beendet und die Summe, die sich im FAC befindet, wird mit FACSTR und STROUT ausgegeben.

Da START(+1) momentan erst auf das Element A(2) weist, ist unsere Routine jedoch noch nicht fertig mit ihrer Aufgabe. A(2) wird mit INFAC in den FAC kopiert, FAC und ARG addiert (JSR ADD, das Ergebnis befindet sich anschließend in FAC) und zum Schleifenanfang gesprungen.

Der Programmablauf ist damit klar. Interessant ist nun ein Zeitvergleich zwischen der Assembler- und einer äquivalenten Basic-Routine. Zuerst muß jedoch das Array initialisiert werden:

```
100 DIM A(2000)
110 FOR I=1 TO 1000:A(I)=I/2:NEXT
120 PRINT"INITIALISIERUNG BEENDET"
130 TA=TI:SYS 49152,A(1),A(1000):TB=TI:PRINT:PRINT"
 ASS.:"(TB-TA)/60
140 TA=TI:FOR I=1 TO 1000:S=S+A(I):NEXT:TB=TI:
PRINT S:PRINT"BASIC:"(TB-TA)/60
```

Initialisiert wird ein Array mit 2000 Elementen, die ersten 1000 Elemente werden mit den Zahlen: 0,5; 1; 1,5; 2;...; 500 besetzt. In Zeile 130 wird die Assemblerroutine aufgerufen und die Zeit mit der internen Uhr TI gestoppt. Anschließend werden die Elemente A(1) bis A(1000) in Basic addiert und dabei ebenfalls die Zeit gemessen.

Wenn ich ehrlich bin, muß ich zugeben, daß das Ergebnis mich selbst verblüfft hat. Die Fließkommaarithmetik ist eine extrem aufwendige Angelegenheit und ich war bisher der Ansicht, daß Assembler bei Verwendung der Fließkommaroutinen keine allzu großen Vorteile gegenüber Basic besitzt, da sich an der langsamen Arbeitsweise dieser Routinen natürlich nichts ändert, wenn sie statt von Basic von einer Maschinenroutine aus aufgerufen werden. Hier das angesichts dieser Tatsache erstaunliche Ergebnis:

1000 Realzahlen addieren:
Basic: 6,25 sec
Assembler: 0,78 sec

Trotz der »gemütlichen« Fließkommaarithmetik ist die Assemblerroutine somit immerhin um den Faktor acht schneller als das vergleichbare Basic-Programm. Extreme

Geschwindigkeitsfreaks können die Routine noch leicht beschleunigen, indem sie den aufwendigen 16-Bit-Vergleich von START(+1) und ENDE(+1) durch einen Zähler ersetzen, der mit der Anzahl der zu summierenden Elemente initialisiert und bei jedem Schleifendurchgang dekrementiert wird. Ein paar weitere Millisekunden pro Schleifendurchgang können eingespart werden, indem JSR KONFAC:RTS durch JMP KONFAC ersetzt wird.

Da es mich reizte, das Programm »Summierung« noch ein wenig auszubauen, stelle ich als »krönenden Abschluß« dieses Artikels das Programm »Statistik« (Listing 12) vor, das die »Varianz« für ein Array innerhalb eines beliebigen Intervalls berechnet.

Für Statistiklaien: Die Varianz ist ein »Streuungsmaß« und gibt Auskunft über die Unterschiedlichkeit der Werte einer Stichprobe. Je höher die Werte, desto weniger einheitlich sind die Stichprobendaten.

$$\text{Varianz} = \left(\sum_{i=1}^{n} (xi)2\right) - \left(\sum_{i=1}^{n} (xi)2\right)/N/N-1$$

Zwei Summen müssen berechnet werden: S1, die Summe aller quadrierten Werte und S2, die Summe aller Werte. Danach wird S2 durch N (N=Anzahl der Werte) geteilt und das Ergebnis von S1 subtrahiert. Das Resultat wird anschließend durch N-1 geteilt.

Nachfolgend ein Basic-Programm, das ein Array mit 1000 Elementen initialisiert, die Varianz sowohl mit der Assemblerroutine als auch in Basic berechnet und die benötigten Zeiten ausgibt:

```
100 DIM A(1000):LG=1:RG=1000:FOR I=LG TO RG:A(I)=
```

Name	Funktion	Vorbereitung	Ergebnis	C64	C16	C128
EINPOS	POS.1-INT.=>FLIESSK.	ÜBERGABE IN Y	RESULT IM FAC	$B3A2	$9A81	
EINNEG	Neg.1-Int.=>FLIESSK.	ÜBERGABE IM AKKU	RESULT IM FAC	$BC3C	$A2C1	
ZWEPOS	Pos.2-Int.=>FLIESSK.	MANT.1/2:H/L;X:$90;SEC	RESULT IM FAC	$BC49	$A2CE	$8C75
ZWENEG	Neg.2-Int.=>FLIESSK.	MANT.1/2:H/L;X:$90;CLC	RESULT IM FAC	$BC44	$A2C9	$8C70
FLIINT	FLIESSK. NACH INT.	FLIESSKOMMAZAHL IM FAC	RES.IN MANTISSE	$BC9B	$A327	$8CC7
FACSTR	FAC IN STRING	FLIESSKOMMAZAHL IM FAC	STRING+POINTER	$BDDD	$A46F	$8E42
STROUT	STRING AUSGEBEN	NACH FACSTR VORHANDEN	AUSGABE	$AB1E	$9088	$55E2
INTOUT	AUSGABE 2BYTE-INT.	X=LOW-BYTE/A=HIGH-BYTE	AUSGABE	$BDCD	$A45F	$8E32
ADD	FAC=ARG+FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$B86A	$9E9E	$8848
SUB	FAC=ARG-FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$B853	$9E87	$8831
MULT	FAC=ARG*FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BA2B	$A07B	$8A27
DIV	FAC=ARG/FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BB12	$A197	$8B4C
EXP	FAC=ARGFAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BF7B	$A5EE	
FAC*10	FAC=FAC*10	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BAE2	$A162	$8B17
FAC/10	FAC=FAC/10	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BAFE	$A183	$8B38
SIN(X)	SINUS-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$E26B	$AA77	
COS(X)	COSINUS-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$E264	$AA70	
SQR(X)	WURZEL-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$BF71	$A5E4	
LOG(X)	LOG.-FKT. (BASIS E)	ARGUMENT X IM FAC	RESULT IM FAC	$B9EA	$A01E	
EXP(X)	EXPONENTIALFUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$BFED	$A660	
FACARG	FAC NACH ARG	FLIESSKOMMAZ. IM FAC	FAC NACH ARG	$BC0C	$A291	
ARGFAC	ARG NACH FAC	FLIESSKOMMAZA. IM ARG	ARG NACH FAC	$BBFC		
FACKON	FAC NACH ADRESSE	POINTER (X=L/Y=H)	FAC NACH ADR	$BBD4	$A24C	
KONFAC	KONSTANTE NACH FAC	POINTER (AKKU=L/Y=H)	KONST. IM FAC	$BBA2	$A21F	
KONADD	FAC=KONSTANTE+FAC	POINTER (AKKU=L/Y=H)	RESULT IM FAC	$B867	$9E9B	
KONMUL	FAC=KONSTANTE*FAC	POINTER (AKKU=L/Y=H)	RESULT IM FAC	$BA28	$A078	
CHKKOM	KOMMA LESEN	KEINE	KEINE	$AEFD	$9491	$795C
GETBYT	1BYTE-INTEGER LESEN	KEINE	X=BYTE	$B79E	$9D84	$87F4
FRMNUM	NUM.AUSDRUCK AUSWERT.	KEINE	RESULT IM FAC	$AD8A	$9314	$FFD7
ADRFOR	FLIESSK.NACH 2-INT.	FLIESSKOMMA IM FAC	Y=LOW/AKKU=HIGH	$B7F7	$9DE4	$8815
GETADR	2BYTE-INTEGER LESEN	KEINE	Y=LOW/AKKU=HIGH			$880F
ADRBYT	2BYTE+1BYTE LESEN	KEINE	$14/15 UND X	$B7EB	$9DD2	$8803
GETPOS	POINTER AUF VARIABLE	KEINE	POINTER (AKKU/Y)	$B08B	$96A5	$7AAF

Tabelle: Vergleichstabelle aller bekannten Funktionen und Adressen für C16, C64, C128

```
I:NEXT:PRINT"START!"
110 :
120 TA=TI:SYS 49152,A(LG),A(RG),V
130 PRINT"V=":V:TB=TI:PRINT"ASS.:"(TB-TA)/60
140 :
150 TA=TI
160 FOR I=LG TO RG
170 S1=S1+A(I)*A(I):S2=S2+A(I)
180 NEXT
190 V=(S1-S2*S2/(RG-LG+1))/(RG-LG)
200 TB=TI
210 PRINT"BASIC:"(TB-TA)/60
```

Die Assemblerroutine benötigt knapp drei Sekunden, die Basic-Routine knapp 16 Sekunden, um die Varianz für 1000 Elemente zu berechnen.

Mit den Kenntnissen, die Sie nun besitzen, und der nachfolgenden Erläuterung von zwei neuen im Programm verwendeten Routinen sollten Sie in der Lage sein, das Programm zu durchschauen und vielleicht auch um die Berechnung weiterer Kennwerte zu erweitern.

Diese zusätzlich verwendeten Routinen vereinfachen vor allem den Umgang mit Konstanten und erlauben es, den FAC direkt mit einer Konstanten zu multiplizieren oder eine Konstante zum FAC zu addieren. Die Routinen erwarten Konstanten, die - ebenso wie Basic-Variablen - im gepackten Fünf-Byte-Format vorliegen (Vorzeichen in Mantisse2 integriert).

Schauen Sie sich bitte in jedem Fall an, wie »Statistik« die errechnete Varianz an das Basic-Programm übergibt. Mit GETPOS wird ein Zeiger auf die im SYS-Befehl angegebene Variable V geholt und mit der neuen Routine FACKON der Inhalt des FAC in diese Basic-Variable kopiert. Komfortabler kann eine Ergebnisübergabe an Basic wohl kaum sein.

KONMUL ($BA28)

KONMUL multipliziert den FAC mit einer Konstanten, auf die in Akku und Y-Register ein Zeiger übergeben werden muß.

KONADD ($B867)

KONPLU addiert FAC und die Konstante, auf die ein ebenfalls in Akku und Y-Register übergebener Zeiger weist.

(Said Baloui/do)




```
ü
 0100            ;*** ZAHLENAUSGABE ***
 0110            ;
 0120            ;AUSGABE EINER FLIESSKOMMAZAHL
 0130            ;AUF DEM BILDSCHIRM.
 0140            ;
 0150            ;1.FLIESSKOMMAZAHL NACH FAC
 0160            ;2.AUFRUF VON FAC=>ASCII
 0170            ;3. AUFRUF VON ASCII=>SCREEN
 0180            ;
 0190FAC         .DE $61
 0200KONFAC      .DE $BBA2
 0210FACSTR      .DE $BDDD
 0220STROUT      .DE $AB1E
 0230            ;
 0240            .BA $C000               ;PROGRAMMSTART
 0250            .OS                     ;CODE GENERIEREN
 0260            ;
 0270            ;*ZAHL NACH FAC UEBERTRAGEN*
 0280            LDA #L,ZAHL
 0290            LDY #H,ZAHL
 0300            JSR KONFAC
 0310            ;
 0320            ;*ZAHL AUSGEBEN*
 0330            JSR FACSTR
 0340            JSR STROUT
 0350            RTS
 0360            ;
 0370            ;*AUSZUGEBENDE ZAHL (PI)*
 0380ZAHL        .BY $82 $49 $0F $DA $A1
 0390            .EN
//

ü
```

Listing 1. Assemblerlisting für Zahlenausgabe

```
ü
 0100            ;*** EIN-BYTE-INTEGER ***
 0110            ;
 0120            ;KONVERTIERUNG EINER EIN-BYTE-
 0130            ;INTEGER-ZAHL INS FLIESSKOMMA-
 0140            ;FORMAT UND AUSGABEE AUF DEM
 0150            ;BILDSCHIRM
 0160            ;
 0170            ;1.INTEGERZAHL NACH FLIESSK.
 0180            ;2.AUFRUF VON FAC=>ASCII
 0190            ;3.AUFRUF VON ASCII=>SCREEN
 0200            ;
 0210EINPOS      .DE $B3A2
 0220EINNEG      .DE $BC3C
 0230FACSTR      .DE $BDDD
 0240STROUT      .DE $AB1E
 0250            ;
 0260            .BA $C000               ;PROGRAMMSTART
 0270            .OS                     ;CODE GENERIEREN
 0280            ;
 0290            ;*POSITIVE INT-ZAHL WANDELN*
 0300            LDY #$80
 0310            JSR EINPOS
 0320            ;
 0330            ;*ZAHL AUSGEBEN*
 0340            JSR FACSTR
 0350            JSR STROUT
 0360            ;
 0370            ;*NEGATIVE INT-ZAHL WANDELN*
 0380            LDA #$80
 0390            JSR EINNEG
 0400            ;
 0410            ;*ZAHL AUSGEBEN*
 0420            JSR FACSTR
 0430            JSR STROUT
 0440            RTS
 0450            ;
 0460            .EN
//
```

Listing 2. Assemblerlisting für Ein-Byte-Integer-Werte

```
ü
 0100            ;*** ZWEI-BYTE-INTEGER ***
 0110            ;
 0120            ;KONVERTIERUNG EINER ZWEI-BYTE-
 0130            ;INTEGER-ZAHL INS FLIESSKOMMA-
```

```
 0140            ;FORMAT UND AUSGABEE AUF DEM
 0150            ;BILDSCHIRM
 0160            ;
 0170            ;1.INTEGERZAHL NACH FLIESSK.
 0180            ;2.AUFRUF VON FAC=>ASCII
 0190            ;3.AUFRUF VON ASCII=>SCREEN
 0200            ;
 0210ZWEIPOS     .DE $BC49
 0220ZWEINEG     .DE $BC44
 0230FACSTR      .DE $BDDD
 0240STROUT      .DE $AB1E
 0250            ;
 0260            .BA $C000               ;PROGRAMMSTART
 0270            .OS                     ;CODE GENERIEREN
 0280            ;
 0290            ;*POSITIVE INT-ZAHL WANDELN*
 0300            LDA #$FF                ;ZAHL: $FFFF
 0310            STA $62                 ;HIGH-BYTE ($FF)
 0320            STA $63                 ;LOW-BYTE ($FF)
 0330            LDX #$90                ;X-REG.: $90
 0340            SEC                     ;CARRY SETZEN
 0350            JSR ZWEIPOS
 0360            ;
 0370            ;*ZAHL AUSGEBEN*
 0380            JSR FACSTR              ;=> AUSGABE VON
 0390            JSR STROUT              ;    65535
 0400            ;
 0410            ;*NEGATIVE INT-ZAHL WANDELN*
 0420            LDA #$FF                ;ZAHL: $FFFF
 0430            STA $62                 ;LOW-BYTE ($FF)
 0440            STA $63                 ;HIGH-BYTE ($FF)
 0450            LDX #$90                ;X-REG.: $90
 0460            CLC                     ;CARRY LOESCHEN
 0470            JSR ZWEINEG
 0480            ;
 0490            ;*ZAHL AUSGEBEN*
 0500            JSR FACSTR              ;=> AUSGABE VON
 0510            JSR STROUT              ;    -1
 0520            RTS
 0530            ;
 0540            .EN
//

ü
```

Listing 3. Assemblerlisting für Zwei-Byte-Integer Werte

```
ü
 0013            ;
 0100            ;*** FLIESSKOMMA ***
 0110            ;
 0120            ;WANDLUNG EINER FLIESSKOMMAZAHL
 0130            ;INS INTEGERFORMAT UND
 0140            ;AUSGABE AUF DEM BILDSCHIRM
 0150            ;
 0160            ;1.FLIESSKOMMAZAHL NACH FAC
 0170            ;2.AUFRUF VON FLIESS=>INTEGER
 0180            ;3.AUFRUF VON FAC=>STRING
 0190            ;4.AUFRUF VON STRING=>SCREEN
 0200            ;
 0210KONFAC      .DE $BBA2
 0220FLIINT      .DE $BC9B
 0230EINPOS      .DE $B3A2
 0240FACSTR      .DE $BDDD
 0250STROUT      .DE $AB1E
 0260            ;
 0270            .BA $C000               ;PROGRAMMSTART
 0280            .OS                     ;CODE GENERIEREN
 0290            ;
 0300            ;*ZAHL NACH FAC UEBERTRAGEN*
 0310            LDA #L,ZAHL
 0320            LDY #H,ZAHL
 0330            JSR KONFAC
 0340            ;
 0350            ;*ZAHL NACH INTEGER WANDELN*
 0360            JSR FLIINT
 0370            ;
 0380            ;*INTEGER NACH FLIESSK. WANDELN*
 0390            LDY $65                 ;INT-ERGEBNIS LADEN
 0400            JSR EINPOS              ;NACH FLIESSK.WANDELN
 0410            ;
 0420            ;
 0430            ;*ZAHL AUSGEBEN*
 0440            JSR FACSTR              ;FAC => STRING
 0450            JSR STROUT              ;STRING => SCREEN
 0460            RTS
 0470            ;
 0480            ;*AUSZUGEBENDE ZAHL (PI)*
 0490ZAHL        .BY $82 $49 $0F $DA $A1
 0500            .EN
//
```

Listing 4. Assemblerlisting für Fließkommaroutinen

```
ü

ü
 0013            ;
 0100            ;*** INTEGERAUSGABE ***
 0110            ;
 0120            ;DIREKTE AUSGABE EINER
 0130            ;POSITIVEN ZWEI-BYTE-ZAHL
 0140            ;AUF DEM BILDSCHIRM
 0150            ;
 0160INTOUT      .DE $BDCD
 0170            ;
 0180            .BA $C000               ;PROGRAMMSTART
 0190            .OS                     ;CODE GENERIEREN
 0200            ;
 0210            ;*AUSGABE VON $F02A (61482)*
 0220            LDX #$2A                ;LOW-BYTE: $2A
 0230            LDA #$F0                ;HIGH-BYTE: $F0
 0240            JSR INTOUT              ;AUSGABE
 0250            RTS
 0260            ;
 0270            .EN
//
```

Listing 5. Assemblerlisting für Integer-Ausgabe

```
ü

ü
 0100            ;*** GRUNDRECHNEN ***
 0110            ;
 0120            ;
 0130ADD         .DE $B86A               ;FAC=ARG+FAC
 0140SUB         .DE $B853               ;FAC=ARG-FAC
 0150MULT        .DE $BA2B               ;FAC=ARG*FAC
 0160DIV         .DE $BB12               ;FAC=ARG/FAC
 0170FAC         .DE $61
 0180ARG         .DE $69
 0190EINPOS      .DE $B3A2
 0200ZWEPOS      .DE $BC49
 0210FACARG      .DE $BC0C
 0220FACSTR      .DE $BDDD
 0230STROUT      .DE $AB1E
 0240BSOUT       .DE $FFD2
 0250            ;
 0260            .BA $C000               ;PROGRAMMSTART
 0270            .OS                     ;CODE GENERIEREN
 0280            ;
 0290            ;*'/': ANTEIL BERECHNEN*
 0300            LDA #L,2000             ;2000,LOW
 0310            STA $63                 ;NACH $63
 0320            LDA #H,2000             ;2000,HIGH
 0330            STA $62                 ;NACH $62
 0340            LDX #$90                ;X INITIALIS.
 0350            SEC
 0360            JSR ZWEPOS              ;2000 => FLIESSK.
 0370            JSR FACARG              ;FAC => ARG
 0380            LDY #3                  ;ANTEIL: 1/3
 0390            JSR EINPOS              ;1/3 => FLIESSK.
 0400            ;
 0410            JSR DIV                 ;FAC=2000/3
 0420            JSR AUSGABE             ;ERGEBNIS AUSGEBEN
 0430            ;
 0440            ;
 0450            ;*'+': 2.FLAECHE ADDIEREN*
 0460            JSR FACARG              ;ERGEBNIS NACH ARG
 0470            LDA #L,5286             ;5286,LOW
 0480            STA $63                 ;NACH $63
 0490            LDA #H,5286             ;5286,HIGH
 0500            STA $62                 ;NACH $62
 0510            LDX #$90                ;X INITIAL.
 0520            SEC
 0530            JSR ZWEPOS              ;5286=>FLIESSK.
 0540            ;
 0550            JSR ADD                 ;FAC=666,66667+5286
 0560            JSR AUSGABE             ;FAC=5952,66667
 0570            ;
 0580            ;
 0590            ;*'-': 52 QM SUBTRAHIEREN*
 0600            JSR FACARG              ;FAC => ARG
 0610            LDY #52                 ;52 NACH
 0620            JSR EINPOS              ;FAC
 0630            ;
 0640            JSR SUB                 ;FAC=5952,66667-52
 0650            JSR AUSGABE             ;FAC=5900,66667
 0660            ;
 0670            ;
 0680            ;*'*': DM-WERT BERECHNEN*
 0690            JSR FACARG              ;FAC => ARG
 0700            LDY #228                ;228 NACH
 0710            JSR EINPOS              ;FAC
```

```
 0720            ;
 0730            JSR MULT              ;FAC=5900,66667*228
 0740            JSR AUSGABE           ;FAC=1345352
 0750            RTS
 0760            ;
 0770            ;
 0780            ;*ZAHL AUSGEBEN*
 0790AUSGABE     JSR FACPUF            ;FAC RETTEN
 0800            JSR FACSTR            ;FAC=>STRING
 0810            JSR STROUT            ;STRING=>SCREEN
 0820            LDA #13               ;ZEILENVORSCHUB
 0830            JSR BSOUT             ;AUSGEBEN
 0840            JSR PUFFAC            ;FAC HOLEN
 0850            RTS
 0860            ;
 0870            ;*FAC RETTEN*
 0880FACPUF      LDX #5
 0890LOOP1       LDA FAC,X
 0900            STA PUFFER,X
 0910            DEX
 0920            BPL LOOP1
 0930            RTS
 0940            ;
 0950            ;*FAC HOLEN*
 0960PUFFAC      LDX #5
 0970LOOP2       LDA PUFFER,X
 0980            STA FAC,X
 0990            DEX
 1000            BPL LOOP2
 1010            RTS
 1020            ;
 1030PUFFER
 1040            .EN
//
```

Listing 6. Assemblerlisting für Grundrechnungsarten

```
 0100            ;*** FUNKTIONEN ***
 0110            ;
 0120            ;
 0130COS         .DE $E264
 0140SIN         .DE $E26B
 0150SQR         .DE $BF71
 0160LOG         .DE $B9EA
 0170EXP         .DE $BFED
 0180FAC         .DE $61
 0190ARG         .DE $69
 0200EINPOS      .DE $B3A2
 0210ZWEPOS      .DE $BC49
 0220FACARG      .DE $BC0C
 0230FACSTR      .DE $BDDD
 0240STROUT      .DE $AB1E
 0250BSOUT       .DE $FFD2
 0260            ;
 0270            .BA $C000             ;PROGRAMMSTART
 0280            .OS                   ;CODE GENERIEREN
 0290            ;
 0300            ;* SIN, COS, SQR, LOG, EXP *
 0310            LDY #10
 0320            JSR EINPOS
 0330            JSR SIN
 0340            JSR AUSGABE
 0350            ;
 0360            LDY #10
 0370            JSR EINPOS
 0380            JSR COS
 0390            JSR AUSGABE
 0400            ;
 0410            LDY #10
 0420            JSR EINPOS
 0430            JSR SQR
 0440            JSR AUSGABE
 0450            ;
 0460            LDY #10
 0470            JSR EINPOS
 0480            JSR LOG
 0490            JSR AUSGABE
 0500            ;
 0510            LDY #10
 0520            JSR EINPOS
 0530            JSR EXP
 0540            JSR AUSGABE
 0550            RTS
 0560            ;
 0570            ;
 0580            ;*ZAHL AUSGEBEN*
 0590AUSGABE     JSR FACSTR            ;FAC=>STRING
 0600            JSR STROUT            ;STRING=>SCREEN
 0610            LDA #13               ;ZEILENVORSCHUB
 0620            JSR BSOUT             ;AUSGEBEN
 0630            RTS
 0640            ;
 0660            .EN
//
```

Listing 7. Assemblerlisting für Funktionen

```
ü
 0100            ;*** USR-VEKTOR ***
 0110            ;
 0120            ;
 0130SQR         .DE $BF71
 0150MULT        .DE $BA2B             ;FAC=ARG*FAC
 0170FAC         .DE $61
 0180ARG         .DE $69
 0210FACARG      .DE $BC0C
 0250            ;
 0260            .BA $C000             ;PROGRAMMSTART
 0270            .OS                   ;CODE GENERIEREN
 0280            ;
 0290            ;*WURZEL(X) BERECHNEN*
 0300            JSR SQR               ;WURZEL(X) BERECHNEN
 0310            JSR FACPUF            ;ERGEBNIS RETTEN
 0320            RTS
 0330            ;
 0340            ;
 0350            .BA $C100             ;START TEIL 2
 0360            ;
 0370            ;*ZAHL(Y) DURCH WURZEL(X) DIVID.*
 0380            JSR FACARG            ;ZAHL(Y) NACH FAC
 0390            JSR PUFFAC            ;WURZEL(X) NACH ARG
 0400            JSR MULT              ;FAC=ARG*FAC
 0410            RTS
 0860            ;
 0870            ;*FAC RETTEN*
 0880FACPUF      LDX #5
 0890LOOP1       LDA FAC,X
 0900            STA PUFFER,X
 0910            DEX
 0920            BPL LOOP1
 0930            RTS
 0940            ;
 0950            ;*FAC HOLEN*
 0960PUFFAC      LDX #5
 0970LOOP2       LDA PUFFER,X
 0980            STA FAC,X
 0990            DEX
 1000            BPL LOOP2
 1010            RTS
 1020            ;
 1030PUFFER
 1040            .EN
//
```

Listing 8. Assemblerlisting für USR-Vektor-Funktionen

```
ü

ü
 0100            ;*** USR-VEKTOR.2 ***
 0110            ;
 0120            ;
 0130ADD         .DE $B86A
 0140FAC         .DE $61
 0150ARG         .DE $69
 0160FACARG      .DE $BC0C
 0170            ;
 0180            .BA $C000             ;PROGRAMMSTART
 0190            .OS                   ;CODE GENERIEREN
 0200            ;
 0210            ;*PARAMETER UEBERNEHMEN*
 0220            DEC 250
 0230            BEQ LOOP3
 0240            JSR FACPUF
 0250            LDA KORR+1
 0260            CLC
 0270            ADC #5
 0280            STA KORR+1
 0290            BCC ENDE
 0300            INC KORR+2
 0310ENDE        RTS
 0320            ;
 0330            ;*RECHNEN*
 0340LOOP3       JSR FACARG
 0350            JSR PUFFAC
 0360            JSR ADD
 0370            LDA LOOP2+1
 0380            CLC
 0390            ADC #5
```

```
 0400            STA LOOP2+1
 0410            BCC SPRUNG
 0420            INC LOOP2+2
 0430SPRUNG      LDA LOOP2+2
 0440            CMP KORR+2
 0450            BNE LOOP3
 0460            LDA LOOP2+1
 0470            CMP KORR+1
 0480            BNE LOOP3
 0490            RTS
 0500            ;
 0510            ;*FAC RETTEN*
 0520FACPUF      LDX #5
 0530LOOP1       LDA FAC,X
 0540KORR        STA PUFFER,X
 0550            DEX
 0560            BPL LOOP1
 0570            RTS
 0580            ;
 0590            ;*FAC HOLEN*
 0600PUFFAC      LDX #5
 0610LOOP2       LDA PUFFER,X
 0620            STA FAC,X
 0630            DEX
 0640            BPL LOOP2
 0650            RTS
 0660            ;
 0670PUFFER
 0680            .EN
//
```

Listing 9. Assemblerlisting für USR-Vektor2-Funktionen

```
ü

ü
 0100            ;*** PARAMETER LESEN ***
 0110            ;
 0120            ;
 0130CHKKOM      .DE $AEFD
 0140GETBYT      .DE $B79E
 0150FRMNUM      .DE $AD8A
 0160ADRFOR      .DE $B7F7
 0170ADRBYT      .DE $B7EB
 0180FACSTR      .DE $BDDD
 0190STROUT      .DE $AB1E
 0200INTOUT      .DE $BDCD
 0210BSOUT       .DE $FFD2
 0220            ;
 0230            .BA $C000               ;PROGRAMMSTART
 0240            .OS                     ;CODE GENERIEREN
 0250            ;
 0260            ;*EIN-BYTE-WERT*
 0270            JSR CHKKOM
 0280            JSR GETBYT              ;EIN-BYTE-WERT
 0290            LDA #0                  ;HIGH-BYTE=0
 0300            JSR INTOUT              ;AUSGEBEN
 0310            LDA #13
 0320            JSR BSOUT
 0330            ;
 0340            ;*ZWEI-BYTE-WERT*
 0350            JSR CHKKOM              ;ZWEI-BYTE-WERT
 0360            JSR FRMNUM              ;NACH FAC
 0370            JSR ADRFOR              ;INS INTEGERFORMAT
 0380            PHA                     ;Y=LOW-BYTE
 0390            TYA                     ;INS
 0400            TAX                     ;X-REGISTER
 0410            PLA                     ;UEBERTRAGEN
 0420            JSR INTOUT              ;AUSGEBEN
 0430            LDA #13
 0440            JSR BSOUT
 0450            ;
 0460            ;*ZWEI- UND EIN-BYTE-WERT*
 0470            JSR CHKKOM              ;ZWEI-BYTE-WERT
 0480            JSR ADRBYT              ;+EIN-BYTE-WERT
 0490            LDA #0                  ;HIGH-BYTE=0
 0500            JSR INTOUT              ;1BYTE AUSGEBEN
 0510            LDA #13
 0520            JSR BSOUT
 0530            LDX $14                 ;X=LOW
 0540            LDA $15                 ;A=HIGH
 0550            JSR INTOUT              ;AUSGEBEN
 0560            LDA #13
 0570            JSR BSOUT
 0580            ;
 0590            ;*BELIEBIGER WERT*
 0600            JSR CHKKOM              ;BELIEBIGEN NUM.
 0610            JSR FRMNUM              ;AUSDRUCK NACH FAC
 0620            JSR FACSTR              ;FAC => STRING
 0630            JSR STROUT              ;STRING AUSGEBEN
 0640            RTS
 0650            ;
 0660            .EN
//
```

Listing 10. Assemblerlisting für Parameterlesen

```
ü

ü
 0100            ;*** SUMMIERUNG ***
 0110            ;
 0120            ;
 0130GETPOS      .DE $B08B               ;=> POINTER
 0140CHKKOM      .DE $AEFD
 0150START       .DE 251                 ;LINKE GRENZE
 0160ENDE        .DE 253                 ;RECHTE GRENZE
 0170ADD         .DE $B86A               ;FAC=ARG+FAC
 0180SUB         .DE $B853               ;FAC=ARG-FAC
 0190MULT        .DE $BA2B               ;FAC=ARG*FAC
 0200DIV         .DE $BB12               ;FAC=ARG/FAC
 0210FAC         .DE $61
 0220ARG         .DE $69
 0230KONFAC      .DE $BBA2
 0240FACARG      .DE $BC0C
 0250FACSTR      .DE $BDDD
 0260STROUT      .DE $AB1E
 0270BSOUT       .DE $FFD2
 0280            ;
 0290            .BA $C000               ;PROGRAMMSTART
 0300            .OS                     ;CODE GENERIEREN
 0310            ;
 0320            ;*POINTER HOLEN*
 0330            JSR CHKKOM              ;POINTER AUF
 0340            JSR GETPOS              ;LINKE GRENZE
 0350            STA START               ;NACH START(+1)
 0360            STY START+1
 0370            JSR CHKKOM              ;POINTER AUF
 0380            JSR GETPOS              ;RECHTE GRENZE
 0390            STA ENDE                ;NACH ENDE(+1)
 0400            STY ENDE+1
 0410            ;
 0420            JSR INFAC               ;A(1) NACH FAC
 0430            ;
 0440BEGIN       JSR FACARG              ;FAC NACH ARG
 0450            LDA START
 0460            CLC                     ;START(+1)=
 0470            ADC #5                  ;START(+1) + 5
 0480            STA START               ;<=>POINTER
 0490            BCC LABEL1              ;AUF NAECHSTE
 0500            INC START+1             ;REALZAHL
 0510            ;
 0520LABEL1      LDA ENDE+1
 0530            CMP START+1
 0540            BCC BACK                ;FERTIG, WENN
 0550            BNE WEITER              ;ENDE(+1)<START(+1)
 0560            LDA ENDE
 0570            CMP START
 0580            BCC BACK
 0590            ;
 0600WEITER      JSR INFAC               ;A(START) => FAC
 0610            JSR ADD                 ;FAC=ARG+FAC
 0620            JMP BEGIN               ;SCHLEIFENANFANG
 0630            ;
 0640BACK        JSR FACSTR              ;AUSGABE VON FAC
 0650            JSR STROUT              ;(=ERGEBNIS)
 0660            RTS
 0670            ;
 0680INFAC       LDA START               ;KONSTANTE, AUF
 0690            LDY START+1             ;DIE START(+1)
 0700            JSR KONFAC              ;WEIST, NACH
 0710            RTS                     ;ARG KOPIEREN
 0720            ;
 0730            .EN
```

Listing 11. Assemblerlisting für Sumierung

```
ü

ü
 0100            ;*** STATISTIK ***
 0110            ;
 0120            ;
 0130GETPOS      .DE $B08B               ;=> POINTER
 0140CHKKOM      .DE $AEFD
 0150SUMME1      .DE $A5
 0160SUMME2      .DE $AA
```

Listing 12. Assemblerlisting für Statistik (Fortsetzung)

```
0170PUFFER      .DE $B2                 ;PUFFER
0180START       .DE $9B                 ;LINKE GRENZE
0190STKOP       .DE $9E                 ;RECHTE GRENZE
0200COUNT       .DE $B7                 ;ZAEHLER
0210N           .DE $B9                 ;ZAEHLER
0220N1          .DE $FA                 ;ZAEHLER
0230ADD         .DE $B86A               ;FAC=ARG+FAC
0240SUB         .DE $B853               ;FAC=ARG-FAC
0250MULT        .DE $BA2B               ;FAC=ARG*FAC
0260DIV         .DE $BB12               ;FAC=ARG/FAC
0270EXP         .DE $BF7B               ;FAC=ARGßFAC
0280FAC         .DE $61
0290ARG         .DE $69
0300KONFAC      .DE $BBA2
0310KONARG      .DE $BA8C
0320FACKON      .DE $BBD4
0330FACARG      .DE $BC0C
0340KONMUL      .DE $BA28
0350KONADD      .DE $B867
0360FLIINT      .DE $BC9B
0370ZWEPOS      .DE $BC49
0380EINPOS      .DE $B3A2
0390FACSTR      .DE $BDDD
0400STROUT      .DE $AB1E
0410            ;
0420            .BA $C000               ;PROGRAMMSTART
0430            .OS                     ;CODE GENERIEREN
0440            ;
0450            ;***POINTER HOLEN***
0460            JSR CHKKOM              ;POINTER AUF
0470            JSR GETPOS              ;LINKE GRENZE
0480            STA START               ;NACH START(+1)
0490            STY START+1
0500            STA STKOP
0510            STY STKOP+1
0520            JSR CHKKOM              ;POINTER AUF
0530            JSR GETPOS              ;RECHTE GRENZE
0540            ;
0550            ;
0560            ;***N, N-1, COUNT ERMITTELN***
0570            SEC
0580            SBC START               ;DIFFERENZ
0590            STA $63                 ;ZWISCHEN DEN
0600            TYA                     ;GRENZEN BERECHNEN
0610            SBC START+1
0620            STA $62
0630            LDX #$90
0640            SEC
0650            JSR ZWEPOS
0660            JSR FACARG
0670            ;
0680            LDY #5                  ;DIFFERENZ GETEILT
0690            JSR EINPOS              ;DURCH 5 ERGIBT
0700            JSR DIV                 ;DIE ANZAHL N-1
0710            LDX #L,N1
0720            LDY #H,N1
0730            JSR FACKON
0740            ;
0750            JSR FLIINT              ;N-1 INKREMENTIEREN
0760            LDX $64                 ;ERGIBT N
0770            LDY $65
0780            INY
0790            BCC NOINC
0800            INX
0810NOINC       STX N+1
0820            STY N
0830            STX COUNT+1             ;UND GLEICHZEITIG
0840            STY COUNT               ;COUNT (COUNT=N)
0850            ;
0860            LDY #0
0870            JSR EINPOS              ;PUFFER INI-
0880            JSR FACPUF              ;TIALISIEREN
0890            ;
0900            ;
0910            ;***SUMME1 BERECHNEN***
0920LOOPS1      JSR INFAC               ;A(START) NACH
0930            LDA START               ;FAC KOPIEREN
0940            LDY START+1             ;UND FAC MIT
0950            JSR KONMUL              ;A(START) MULTIPL
0960            LDA #L,PUFFER           ;FAC UND SUMME1
0970            LDY #H,PUFFER           ;ADDIEREN UND
0980            JSR KONADD              ;NEUE SUMME1 IN
0990            JSR FACPUF              ;PUFFER KOPIEREN
1000            JSR OFFSET              ;START INKREM.
1010            JSR DECCNT              ;ZAEHLER DEKREM.
1020            BNE LOOPS1              ;FERTIG? NEIN=>
1030            ;
1040            LDA STKOP               ;START WIEDER
1050            LDX STKOP+1             ;AUF DAS ERSTE
1060            STA START               ;ELEMENT A(LG)
1070            STX START+1             ;SETZEN
1080            LDA N
1090            LDX N+1                 ;COUNT MIT DER
1100            STA COUNT               ;ANZAHL N LADEN
1110            STX COUNT+1
1120            LDY #0                  ;UND FAC
1130            JSR EINPOS              ;INITIALSIEREN
1140            ;
1150            ;
1160            ;*** SUMME2 BERECHNEN ***
1170LOOPS2      LDA START               ;ZU FAC DAS ELE-
1180            LDY START+1             ;MENT A(START)
1190            JSR KONADD              ;ADDIEREN
1200            JSR OFFSET              ;START INKREM.
1210            JSR DECCNT              ;ZAEHLER DEKREM.
1220            BNE LOOPS2              ;FERTIG? NEIN=>
1230            ;
1240            ;
1250            ;*** SUMME2=SUMME2*SUMME2 ***
1260            JSR FACARG              ;SUMME2 NACH ARG
1270            LDY #2                  ;ZAHL 2 NACH FAC
1280            JSR EINPOS
1290            JSR EXP                 ;FAC=FACßARG
1300            ;
1310            ;
1320            ;*** SUMME2 DURCH N DIVID. ***
1330            JSR FACARG              ;SUMME2 NACH ARG
1340            LDA N                   ;BRINGEN
1350            LDX N+1
1360            STA $63                 ;N NACH FAC
1370            STX $62                 ;BRINGEN
1380            LDX #$90
1390            SEC
1400            JSR ZWEPOS              ;SUMME2 DURCH
1410            JSR DIV                 ;N DIVIDIEREN
1420            ;
1430            ;
1440            ;*** SUMME2 VON SUMME1 SUB. ***
1450            LDA #L,PUFFER           ;SUMME1 NACH ARG
1460            LDY #H,PUFFER           ;BRINGEN UND
1470            JSR KONARG              ;SUMME2 VON
1480            JSR SUB                 ;SUMME1 SUBTRAH.
1490            ;
1500            ;
1510            ;*** RESULTAT DURCH N-1 DIVID.***
1520            JSR FACARG              ;RESULTAT NACH
1530            LDA #L,N1               ;ARG BRINGEN
1540            LDY #H,N1               ;KONSTANTE N NACH
1550            JSR KONFAC              ;FAC UND ARG
1560            JSR DIV                 ;DURCH FAC DIV.
1570            ;
1580            ;
1590            ;*** VARIANZ IN V SPEICHERN ***
1600            JSR CHKKOM              ;ZEIGER AUF V
1610            JSR GETPOS              ;HOLEN UND
1620            TAX                     ;DIE BERECHNETE
1630            JMP FACKON              ;VARIANZ NACH V
1640            ;
1650            ;
1660            ;* ZAEHLER DEKREMENTIEREN *
1670DECCNT      LDA COUNT
1680            BNE LAB1
1690            DEC COUNT+1
1700LAB1        DEC COUNT
1710            LDA COUNT+1
1720            BNE DECEND
1730            LDA COUNT
1740DECEND      RTS
1750            ;
1760            ;
1770            ;* A(START) NACH FAC *
1780INFAC       LDA START               ;KONSTANTE, AUF
1790            LDY START+1             ;DIE START(+1)
1800            JMP KONFAC              ;WEIST, NACH
1810            ;
1820            ;* KONST.IN PUFFER NACH FAC *
1830PUFFAC      LDA #L,PUFFER
1840            LDY #H,PUFFER
1850            JMP KONFAC
1860            ;
1870            ;* FAC NACH PUFFER KOPIEREN *
1880FACPUF      LDX #L,PUFFER
1890            LDY #H,PUFFER
1900            JMP FACKON
1910            ;
1920            ;* START AUF NEXT ELEMENT *
1930OFFSET      LDA START
1940            CLC
1950            ADC #5
1960            STA START
1970            BCC OFFEND
1980            INC START+1
1990OFFEND      RTS
2000            ;
2010            .EN
//
```

Listing 12. Assemblerlisting für Statistik (Schluß)