Von Basic zu Assembler (Teil 11)

Tabellen können Assembler-Programme erheblich schneller machen! Wie man mit den verschiedenen Tabellensorten umgeht, zeigt Ihnen diese Folge. C 64- und C 128-Benutzer erhalten ein Programm, das Ihnen die Umwandlung beliebiger Zahlen in die beiden Computerformate FLPT und MFLPT abnimmt.

Nachdem wir in der letzten Folge die Fließkommazahlen kennengelernt und erfahren haben, wie unser Computer diese Zahlen speichert und verarbeitet, sollen Sie in dieser Folge zunächst einmal je ein Programm für den C 64 und den C 128 (in Basic) präsentiert bekommen, das Ihnen die aufwendige Arbeit des »zu Fuß«-Berechnens der FLPT- und MFLPT-Formate beliebiger Zahlen erspart:

Beide Programme (Listing 1 und 2) verwenden den sogenannten programmierten Direktmodus und steuern damit einen Maschinensprachemonitor an (für den C 64 muß man vor dem Start noch den SMON geladen haben!). Dabei läuft das C 128-Programm automatisch, beim C 64-Programm ist es noch nötig, nach der Monitormeldung viermal <RETURN> zu drücken (SMON scheint den Tastaturpuffer nicht in gewohnter Weise zu behandeln). Auf dem Bildschirm erscheint dann die Einschaltmeldung des Monitors. Nach Druck auf <RETURN> sehen Sie die Speicherbereiche ab $6000 und $6010. In diese Bereiche transportierte ein kleines Maschinenprogramm die zuvor eingegebene Zahl als MFLPT- (ab $6000) und als FLPT-Zahl (ab $6010). Das Maschinenprogramm findet sich in den DATA-Zeilen des Listings und ist dabei in REM-Zeilen als Quelltext dargestellt. Der Sprung in die Routine $BBD4 (genannt MOVMF) ist C 64-Benutzern schon aus den letzten Folgen vertraut: Die Register X und Y weisen als Zeiger auf eine Speicherstelle, in die durch MOVMF der Inhalt des FAC unter gleichzeitiger Umwandlung ins MFLPT-Format transportiert wird. C 128-Benutzer finden diese Routine ab Adresse $8C03. Ihre Funktionsweise unterscheidet sich nicht von der entsprechenden C 64-Routine.

Fließkommazahlen per USR übergeben

Wie haben wir die Zahlen übrigens in den FAC hineinbekommen? Da gibt es das — vom Basic-Programmierer gemiedene — Kommando USR(n), wobei n ein beliebiges Argument sein kann. Dieses n nun findet man nach dem USR-Kommando im FAC vor. Wie funktioniert USR? Stößt der Interpreter auf dieses Kommando, dann führt er einen Sprung in ein Maschinenprogramm aus, dessen Adresse als Vektor beim C 64 in den Speicherzellen $311/$312 (dezimal 785/786) gespeichert ist. Er weist im allgemeinen auf die Adresse $B248, wo die Ausgabe der Fehlermeldung »SYNTAX ERROR« ausgegeben und ein Programmabbruch ausgeführt wird. Der C 128 versteckt diesen Vektor in den Speicherstellen $1219/$121A (dezimal 4633/4634). Sein Inhalt zeigt normalerweise auf die Adresse $7D28, die den »ILLEGAL QUANTITY ERROR« behandelt.

In unseren beiden Programmen verbiegen wir einfach diese USR-Vektoren, so daß sie auf $1600 (C 128) oder $334 (C 64) zeigen, wohin wir unsere kleine Assembler-Routine gelegt haben. Der USR-Aufruf schaltet in dieses kleine Programm und transportiert das Argument n in den FAC. Wir könnten durch das M-Kommando des Monitors auch direkt in den FAC hineinsehen, würden dort aber nicht mehr unsere Zahl entdecken. Der FAC wird vom Zeitpunkt des USR-Aufrufes bis zur Ausführung des M-Kommandos verändert. Deshalb die Verschiebung des FAC-Inhaltes nach $6010.

Das USR-Kommando ist zweifellos die bequemste Methode, Fließkommazahlen von Basic aus an ein Assemblerprogramm zu übergeben. Leider ist das aber nur für einen Wert einfach. Werden es mehrere, dann steigt der Programmaufwand. Eine andere Methode haben wir in den letzten Folgen kennengelernt, nämlich die Übergabewerte durch FRMNUM aus dem Basic-Text zu lesen. Eine weitere Methode lernen wir in der kommenden Folge kennen: Variable werden vom Basic-Interpreter in einer Tabelle abgelegt, die man durchaus auch von Assemblerprogrammen her benutzen kann. Bevor wir uns aber diesen Möglichkeiten zuwenden, werden wir diesmal noch etwas mehr über Tabellen erfahren.

Tabellen

Zur Ausrüstung von Schülern und Studenten (und vielen anderen) gehörte früher auch ein ständig mitgeschlepptes Tabellenwerk, in dem sich dann beispielsweise die Logarithmen der Zahlen von 1 bis 1000 fanden oder die Sinuswerte der Winkel von 0 bis 90 Grad und vieles andere mehr. Dann kam die Revolution durch die Taschenrechner: Kein mühseliges Nachschlagen mehr, kein Interpolieren, hohe Genauigkeit! Der Computer hat die Tabellen verdrängt ... oder doch nicht?

Zwar werden solche Arbeiten wie das Berechnen eines Sinus oder von Logarithmen im Computer durch Entwicklung von Potenzreihen erledigt. Das dauert aber verhältnismäßig lange und für besonders zeitkritische Programme greift der Assembler-Programmierer auf Tabellen zurück! Wir finden Tabellen in unserem Computer in verschiedenen Erscheinungsformen: als Zahlentabellen mit Integer- oder Fließkommawerten, als Texttabellen, als Adressentabellen und als Sprungtabellen.

Tabellen im ROM

Falls Sie mal in der Situation sein sollten, beispielsweise den Wert 2*Pi in einem Programm benutzen zu müssen, dann können Sie sich viel Rechnerei ersparen, mit der Sie diese Zahl in das MFLPT-Format bringen: Im ROM befindet sich 2*Pi nämlich schon abrufbereit, genauso wie eine ganze Reihe weiterer Zahlen und Texte. Die ROM-Bereiche unseres Computers liefern uns also nicht nur Assembler-Routinen, die wir ansteuern. Sie sind auch eine Datenquelle. Damit Sie wissen, wo Sie was im Computer finden können, sehen Sie sich die auf Seite 140 abgedruckten Werte der Tabelle an.

Tabelle 1 listet die wichtigsten Zahlentabellen im ROM des C 64 auf.

Die Tabelle 2 zeigt die Zahlentabellen des C 128.

Die Tabellen 3 und 4 beziehen sich auf die Texttabellen im ROM des C 64 und des C 128.

Einige weitere interessante Tabellen im ROM des C 128 listet Tabelle 5 auf.

Schließlich finden Sie in Tabelle 6 noch die Sprungtabelle im C 128 und ihre Zuordnungen.

Außer den hier vorgestellten Tabellen finden sich natürlich noch weitere in den ROM-Bausteinen: Da gibt es Tabellen zur Decodierung der Tastatur, Tabellen von Farbwerten, Tabellen zur Initialisierung des Systems, die Default-Werte (Einschaltwerte) enthalten und so weiter.

Eigene Tabellen

Interessanter als die eben behandelten ROM-Tabellen sind natürlich Tabellen in eigenen Programmen. Nehmen wir einmal an, Sie benötigen in einem Programm sehr häufig irgendwelche Potenzen von 2 (also 2 hoch 3, 2 hoch 4 und so weiter). Die dabei vorkommenden Hochzahlen bewegen sich zwischen 0 und 7. Nun können Sie natürlich jedesmal den Potenzwert ausrechnen, beispielsweise bei der Zahl 2 hoch 5:

```
LDA #$02 ;Basis in den Akku
          laden, also 2
ASL      ;mal 2
ASL      ;mal 2
ASL      ;mal 2
ASL      ;mal 2
```

Nun steht das Ergebnis im Akku und Sie können damit weiter operieren. Komplizierter wird das aber schon, wenn Sie nicht Potenzen von 2, sondern — sagen wir mal — von 3 oder 5 benötigen. Besser geht das und meist auch schneller mit Tabellen. Wir legen irgendwo eine Tabelle der Potenzen von 2 an:

```
TAB 1,2,4,8,16,32,64,128
```

Brauchen wir nun 2 hoch 5, dann schieben wir die Hochzahl in ein Indexregister und laden den Akku durch die indizierte Adressierung:

```
LDX #$05  ;Das ist die
           Hochzahl
LDA TAB,X ;und schon ist
           32 im Akku!
```

Es spielt nun auch keine Rolle mehr, ob wir die Potenzen der Zahl 2, 3 oder irgendeiner anderen Basiszahl benötigen: Tabelle anlegen, Hochzahl als Index wählen und den Akku indiziert laden. Braucht man für andere Zwecke aufeinanderfolgende Elemente der Tabelle, dann ge-

nügt es nun, durch INX oder DEX den Index zu variieren.

Komplexe Tabellen

Diese einfachste Art der Ansteuerung einer Tabelle hat natürlich gewisse Einschränkungen zur Folge: Die Elemente dürfen nicht größer als 255 (also 1 Byte) sein, es dürfen nicht mehr als 256 Elemente verwendet werden.

Hätte unsere Potenztabelle nun immer 16-Bit-Werte aufgelistet, gehörten also zu jedem Element 2 Byte, dann müßte der Index vor dem Zugriff in die Tabelle jeweils verdoppelt werden. Dazu wieder unsere Tabelle der Zweierpotenzen als Beispiel:

```
0.1, 0.2, 0.4, 0.8,
0.16, 0.32, 0.64, 0.128
```

Hier haben wir die Potenzwerte jeweils in der Reihenfolge MSB,LSB abgelegt. Suchen wir nun den Wert für 2 hoch 5, dann programmieren wir:

```
LDA #$05  ;Das ist wieder
          die Hochzahl
ASL       ;verdoppeln
TXA       ;und ins X-Regi-
          ster schieben
LDA TAB,X ;laden des MSB
          (10. Wert in der
          Tabelle)
STA ...   ;und ablegen an
          der Stelle, an der
          es gebraucht wird
INX       ;Index erhöhen
LDA TAB,X ;laden des LSB
```

Damit hätten wir dann die 16-Bit-Zahl aus der Tabelle gelesen. Anstelle der beiden letzten Zeilen hätte auch eine einzige genügt:

```
LDA TAB+1,X ;laden des LSB
```

Adressen sind solche 16-Bit-Werte und daher findet man diese Technik der Tabellenmanipulation auch sehr häufig bei Adressentabellen. Beispielsweise haben wir im ersten Modul des Programms 3 (Folge 7, Ausgabe 10/86, Seite 156) ab Zeile 970 auf diese Weise eine Sprungadresse aus der Tabelle SPRTAB gelesen.

Es gibt Tabellen, deren Elemente jeweils mehr als 2 Byte enthalten. In solchen Fällen genügen häufig zwei oder mehrere ASL des Index oder aber man führt jeweils eine Addition des entsprechenden Offset zum Index aus.

Lange Tabellen

Einige Tabellen, besonders Texttabellen, sind länger als 256 Byte. In dem Fall ist es nicht mehr möglich, die einzelnen Elemente (oder Teile der Elemente) mittels der bisher angewandten absolut X-indizierten (oder auch Y-indizierten) Adressierung anzusprechen, denn die Register fassen nur Zahlen von 0 bis 255. Wir greifen dann zur indirekt-indizierten Adressierung. Ein 16-Bit-Zeiger in der Zeropage wird mit der Startadresse der Tabelle geladen, das Y-Register dient als Index. Das Ansprechen der einzelnen Bytes geschieht dann beispielsweise wie folgt:

```
LDA  INDEX  ;aktuellen Index laden
ASL         ;und verdoppeln (Elemente sind 2-Byte-
             Werte)
BCC  KLEIN  ;verzweigen, wenn dabei kein Überlauf
             eintrat
INC  ZERO+1 ;bei Überlauf MSB der Tabellen-Startadresse
             erhöhen
TAY         ;Offset ins Indexregister schieben
LDA (ZERO),Y;in ZERO und ZERO+1 liegt die Startadresse
             der Tabelle
            ;und wir haben das LSB eines Elementes
             geladen
STA ...     ;an geeigneter Stelle speichern
INY         ;Indexregister auf MSB richten
LDA (ZERO),Y;MSB laden
STA ...     ;und an geeigneter Stelle weiterverwenden
```

Dabei war ZERO/ZERO+1 der Vektor in der Zeropage, der auf den Tabellenstart wies und INDEX eine Speicherstelle, die den gerade aktuellen Index enthielt, beispielsweise die Hochzahl bei einer Potenztabelle. Noch mehr Möglichkeiten bieten sich, wenn man für den Index einen 16-Bit-Wert reserviert. Im folgenden Beispiel seien INDEX/INDEX+1 die dafür gedachten Speicherstellen:

```
LDA  INDEX    ;LSB des Index laden
ASL           ;und verdoppeln (Elemente sind 2-Byte-
               Werte)
TAY           ;Offset ins Indexregister schieben
LDA  INDEX+1  ;MSB des Index laden
ROL           ;Ebenfalls verdoppeln, aber mit Carry-Bit
ADC  ZERO+1   ;dazu MSB der Tabellenadresse addieren
STA  ZERO+1   ;und als neues MSB merken
LDA (ZERO),Y  ;jetzt LSB des aktuellen Elementes laden
STA  ...
INY           ;Indexregister auf MSB richten
LDA (ZERO),Y  ;und MSB des Elementes laden
STA  ...
```

Auf diese oder ähnliche Weise können Sie noch so ausgedehnte Tabellen beherrschen.

Texttabellen

Im Vergleich zu Zahlen- oder Adreßtabellen weisen Texttabellen meist die Besonderheit von Elementen variabler Bytezahl auf. Beim Lesen der einzelnen Bytes eines Elements fügt man hier immer eine Prüfung auf ein Textende-Merkmal ein. Solche Merkmale sind beispielsweise Nullbytes. Durch ein BEQ kann dann reagiert werden und zwei Nullbytes markieren das Ende der Tabelle. Manchmal verwendet man auch etwas platzsparendere Kennzeichen wie ein gesetztes Bit 7 eines Zeichens. Dann darf allerdings die Tabelle keine Zeichen enthalten, die von sich aus schon mit gesetzten Bit 7 aufwarten. Hier wird dann durch BMI oder BIT und nachfolgendes Abfragen der entsprechenden Flaggen geprüft, ob ein Textende-Merkmal vorliegt.

Adressentabellen

Das Lesen von Adressentabellen haben wir vorhin bei den Zahlentabellen schon mitbehandelt. Sie verhalten sich wie Tabellen mit 2-Byte-Elementen. Hier soll es nun darum gehen, wie man die so gefundenen Adressen weiterverwendet, um einen Sprung an die herausgesuchte Adresse zu vollziehen. Die Technik der selbstmodifizierenden Sprunganweisung haben wir im Programm 3 (Folge 7, Ausgabe 10/86, Seite 156) in Zeile 1070 gewählt. Die Zeilen 1000 bis 1030 lesen LSB und MSB der Zieladresse aus der Sprungtabelle und tragen sie hinter die JSR-Anweisung in Zeile 1070 ein. Dorthin gelangt danach das Programm und vollzieht den Sprung.

Der Nachteil dieser Technik ist, daß sie nur in RAM-Bereichen funktioniert, weil ins Programm geschrieben werden muß. Arbeitet man mit ROMs oder EPROMs, dann bieten sich zwei andere Möglichkeiten an, von denen wir zuerst die Verwendung eines indirekten Sprungs vorstellen wollen. Dazu speichert man die gelesenen Tabellenwerte in einen Vektor aus der Zeropage (beispielsweise ZWSP/ZWSP+1) und springt dann mit

```
JMP (ZWSP) ;das ist der selten
            benutzte indirekte
            Sprung
```

in die gesuchte Routine. Nebenbei bemerkt: ZWSP/ZWSP+1 muß nicht unbedingt in der Zeropage stehen: Man kann beliebige andere Speicherbereiche für diesen Vektor verwenden.

Auf den ersten Blick etwas irritierend wirkt die andere Technik, die sich des Stapels bedient. Hier ein Beispiel:

```
LDA INDEX ;aktuellen Index
          laden
ASL       ;und verdoppeln
          (Adresstabelle!)
TAX       ;ins Indexregi-
          ster schieben
INX       ;Indexregister
          auf MSB richten
LDA TAB,X ;MSB der Ziel-
          adresse laden
PHA       ;und auf den
          Stapel schieben
DEX       ;Indexregister
          auf LSB richten
LDA TAB,X ;LSB der Ziel-
          adresse laden
PHA       ;und auf den
          Stapel schieben
RTS       ;!!!
```

Die Frage ist: Was macht RTS? Hier die Antwort und gleichzeitig die Lösung des Rätsels:

1) RTS holt die auf dem Stapel gespeicherte Adresse ab und schreibt sie in den Programmzähler. Damit die Reihenfolge LSB/MSB stimmt, muß als letztes das LSB im Stapel landen.

2) RTS vermindert dann den Stapelzeiger um 2. Das sei nur der Vollständigkeit halber gesagt.

3) RTS addiert zum Programmzähler eine 1 und dann läuft das Programm von dieser Adresse an weiter.

Insgesamt ergibt sich daraus dann ein Sprung zum gewünschten Programm. Wegen des dritten Punktes der RTS-Tätigkeit muß man aber darauf achten, daß in der Adressentabelle nicht ZIELADRESSE, sondern immer ZIELADRESSE-1 steht!

Mir wird bei diesem Sprung über den Stapel immer etwas mulmig zumute. Allzu unklar ist der Gebrauch des RTS. Ich bin mir nie so ganz sicher, ob ich (oder ein anderer Benutzer) nach einigen Monaten ein Programm mit diesem Trick noch völlig durchschauen kann.

Ich verweise zum Schluß noch auf einen Artikel von Florian Müller: Effektives Programmieren in Assembler (erschienen im Sonderheft 8/85, Seite 74). In den Kapiteln 5 und 6 werden dort einige Beispiele zur Tabellenverwendung gezeigt.

(Heimo Ponnath/dm)

Startadresse($)	Format	Inhalt
AEA8	MFLPT	Pi
B1A5	MFLPT	-32768
B9BC	MFLPT	1
B9C2	MFLPT	Polynomkoeffizienten für LOG-Berechnung
B9D6	MFLPT	SQR(1/2)
B9DB	MFLPT	SQR(2)
B9E0	MFLPT	-0.5
B9E5	MFLPT	ln 2
BAF9	MFLPT	10
BDB3	MFLPT	99 999 999.9
BDB8	MFLPT	999 999 999
BDBD	MFLPT	1 000 000 000
BF11	MFLPT	0.5
BFBF	MFLPT	1/ln2
BFC5	MFLPT	Polynomkoeffizienten für EXP-Berechnung
BFE3	MFLPT	ln 2
BFE8	MFLPT	1
E2E0	MFLPT	Pi/2
E2E5	MFLPT	2*Pi
E2EA	MFLPT	0.25
E2F0	MFLPT	Konstanten für die Entwicklung von SIN,COS,...
E309	MFLPT	2*Pi
E33F	MFLPT	Konstanten für die Entwicklung von ATN
E376	MFLPT	1

Tabelle 1. Die wichtigsten Zahlentabellen im ROM des C 64

Startadresse($)	Label	Format	Inhalt
69D8	n320	MFLPT	320*65535
69DD	n200	MFLPT	200*65535
6FF9	scalel	1-Byte	LSB der Frequenzen
7005	scaleh	1-Byte	MSB der Frequenzen danach weitere Tabellen mit Werten zur Musikprogrammierung
849A	n32768	MFLPT	-32768
899C	fone	MFLPT 1	
89A2	logco3	MFLPT	Koeffizienten für LOG-Berechnung
89B6	sqr05	MFLPT	SQR(1/2)
89BB	sqr20	MFLPT	SQR(2)
89C0	neghlf	MFLPT	-0.5
89C5	log2	MFLPT	ln 2
8B2E	tenc	MFLPT	10
8E17	n0999	MFLPT	99 999 999.9
8E1C	n9999	MFLPT	999 999 999
8E21	nmil	MFLPT	1 000 000 000
8F76	fhalf	MFLPT	0.5
9005	logeb2	MFLPT	1/ln2
900B	expco7	MFLPT	Koeffizienten für EXP-Berechnung
9485	pi2	MFLPT	Pi/2
948A	twopi	MFLPT	2*Pi
948F	fr4	MFLPT	0.25
9495	sinco5	MFLPT	Koeffizienten für SIN,COS,...
94AE	sinco0	MFLPT	2*Pi
94E4	atnc11	MFLPT	Koeffizienten für ATN-Berechnung
951B	atnc00	MFLPT	1
9F29	angval	2-Byte	Sinuswerte 0 bis 90 Grad in 10 Grad-Schritten

Tabelle 2. Die wichtigsten Zahlentabellen im C 128-ROM

Startadresse($)	Inhalt
A004	CBMBASIC
A09E	Texte der Basic-Befehlsworte (im letzten Byte ist jeweils das Bit 7 gesetzt)
A19E	Texte der Basic-Fehler- und Systemmeldungen (im letzten Byte ist Bit 7 gesetzt)
A364	Weitere Systemmeldungen:OK,ERROR,... (das letzte Byte ist jeweils 0)
ACFC	Fehlermeldungen für INPUT (letztes Byte ist 0)
E460	BASIC BYTES FREE
E473	Einschaltmeldung
ECE6	LOAD <RETURN>,RUN <RETURN>
F0BD	Texte für Ein- und Ausgabe-Operationen
FD10	CBM80

Tabelle 3. Die wichtigsten Texttabellen im C 64-ROM

Quelle: Schineis, Braun, Grellner, »C 128 ROM-Listing Basic 7.0-Betriebssystem«, Markt & Technik Verlag, München 1986

Fortsetzung auf Seite 142

Startadresse($)	Label	Inhalt
41BB	sigmsg	Systemmeldung bei Kaltstart
4417	reslst	Liste der Basic-Befehlsworte (Bit 7 des letzten Byte ist jeweils gesetzt)
484B	errtab	Liste der Fehlermeldungen (Bit 7 des letzten Byte ist jeweils gesetzt)
63F5		Namen der Programmautoren
A7E8		ARE YOU SURE?
CEB2	pky1	Standardtexte der Funktionstasten
F6B0	msgtbl	Kernel-Textmeldungen
F90B		BOOTING

Tabelle 4. Die wichtigsten Texttabellen im ROM des C 128

Startadresse($)	Label	Inhalt
46FC	stmdsp	Adressentabelle der Basic-Befehle
AE63	kydmsg	verschlüsselte Mitteilung der Programmautoren
AF00	jmptbl	Sprungtabelle der Interpreter-Routinen
C6DD	funtab	ASCII-Codes der Funktionstasten
CE74	loczp	Tabelle der Default-Werte 40-Zeichen-Bildschirm
CE8E	locabs	Tabelle der Default-Werte 80-Zeichen-Bildschirm
F7F0	config	MMU-Konfigurationen für BANK 0 bis BANK 15
FF47	kspio	Kernel-Sprungtabelle
FFF8	system	Tabelle der Systemvektoren (Initialisierung, NMI, Reset und IRQ)

Tabelle 5. Einige andere wichtige Tabellen im ROM des C 128

Inhalt		Ziellabel	Funktion
JMP	$84B4	ayint	FAC —> Integer mit Vorzeichen
JMP	$793C	givayv	Integer in Y/A zu FLPT in FAC
JMP	$8E42	fout FAC	—> String, Adresse in A/Y
JMP	$8052	val1	String auswerten
JMP	$8815	getadr	FAC —> Integer in Y/A
JMP	$8C75	floatc	Exponent in FAC, normalisieren
JMP	$882E	fsub	FAC = FAC - (A/Y)
JMP	$8831	fsubt	Basic-Funktion Minus
JMP	$8845	fadd	FAC = FAC + (A/Y)
JMP	$88[illegible]3	faddt	Basic-Funktion Plus
JMP	$8[illegible]24	fmult	FAC = FAC * (A/Y)
JMP	$8A27	fmultt	Basic-Funktion Mal
JMP	$8B49	fdiv	FAC = (A/Y) / FAC
JMP	$8B4C	fdivt	Basic-Funktion Division
JMP	$89CA	log	Basic-Funktion LOG
JMP	$8CFB	int	Basic-Funktion INT
JMP	$8FB7	sqr	Basic-Funktion SQR
JMP	$8FFA	negop	Basic-Funktion negatives Vorzeichen
JMP	$8FBE	fpwr	
JMP	$8FC1	fpwrt	Basic-Funktion Potenz
JMP	$9033	exp	Basic-Funktion EXP
JMP	$9409	cos	Basic-Funktion COS
JMP	$9410	sin	Basic-Funktion SIN
JMP	$9459	tan	Basic-Funktion TAN
JMP	$94B3	atn	Basic-Funktion ATN
JMP	$8C47	round	FAC runden
JMP	$8C84	abs	Basic-Funktion ABS
JMP	$8C57	sign	Vorzeichenflag —> Akku
JMP	$8C87	fcomp	FAC mit (A/Y) vergleichen
JMP	$8437	rnd0	Zufallszahl holen
JMP	$8AB4	conupk	(A/Y) —> FAC
JMP	$8A89	romupk	(A/Y) —> ARG
JMP	$7A85	movfrm	(A/Y) —> FAC
JMP	$8BD4	movfm	(A/Y) —> FAC
JMP	$8C00	movmf	FAC —> (X/Y)
JMP	$8C28	movfa	ARG —> FAC
JMP	$8C38	movaf	FAC —> ARG
JMP	$4828	optab	Tabelle der Prioritätsflags der mathematischen Routinen
JMP	$9B30	drawln	Strecke zeichnen
JMP	$9BFB	gplot	Punkt setzen
JMP	$6750	cirsub	Drehung ausführen
JMP	$5A9B	run	Basic-Statement RUN
JMP	$51F3	runc	Basic-Zeiger initialisieren, CLR
JMP	$51F8	clear	Basic-Statement CLR
JMP	$51D6	new	Basic-Statement NEW
JMP	$4F4F	lnlprg	berechnen der Linkadressen
JMP	$430A	crunch	Wandlung von Text in Tokens
JMP	$5064	fndlin	
JMP	$4AF6	newstt	Stoptaste abfragen, nächsten Basic-Befehl holen
JMP	$78D7	eval	Ausdruck auswerten
JMP	$77EF	frmevl	folgenden Ausdruck auswerten
JMP	$5AA6	runprg	aktives Programm starten
JMP	$5A81	setexc	Programm-Modus setzen
JMP	$50A0	linget	Zeilennummer holen
JMP	$92EA	garba2	Garbage collection ausführen
JMP	$4DCD	execln	

Tabelle 6. Die C 128-Sprungtabelle der Interpreter-Routinen

```
10 REM ************************************
   *************                          <054>
20 REM *                                  
                *                          <069>
30 REM *   PROGRAMM ZUM UMWANDELN VON ZAHL
   EN IN DIE    *                         <155>
40 REM *   C 64-FORMATE    MFLPT   (AB $6000
   )            *                         <059>
50 REM *                    FLPT   (AB $6010
   )            *                         <010>
60 REM *                                  
                *                          <109>
70 REM *        HEIMO PONNATH  HAMBURG  198
   6            *                         <199>
80 REM *                                  
                *                          <129>
90 REM ************************************
   *************                          <134>
100 REM                                   <162>
110 PRINT CHR$(147)"IST DER SMON AB $C000
    SCHON EINGELADEN{2SPACE}(J/N)";:INPUT
    A$                                    <043>
120 IF A$<>"J" THEN PRINT"WUENSCHE GUTEN A
    BSTURZ...ODER SMON LADEN!":END        <020>
130 FOR I=0 TO 17:REM EINLESEN DES MASCHIN
    ENPROGRAMMES                          <001>
140 READ D:POKE 828+I,D                   <223>
150 NEXT I                                <234>
160 REM -------- DAS MASCHINENPROGRAMM ---
    ---------                             <116>
170 DATA 162,000    :REM LDX #$00     ;LSB
     ZIELADRESSE                          <111>
180 DATA 160,096    :REM LDY #$60     ;MSB
       -"-                                <028>
190 DATA 032,212,187:REM JSR $BBD4    ;FAC
    -> (X/Y)                              <161>
200 DATA 162,006    :REM LDX #$06     ;ZAE
    HLER EINRICHTEN                       <165>
210 DATA 181,096    :REM LDA $60,X    ;FAC
     AUSLESEN                             <048>
220 DATA 157,015,096:REM STA $600F,X  ;UND
     UEBERTRAGEN                          <156>
230 DATA 202        :REM DEX          ;ZAE
    HLER -1                               <008>
240 DATA 208,248    :REM BNE $033D    ;WEI
    TER BIS FAC UEBERTRAGEN IST           <236>
250 DATA 096        :REM RTS          ;ZUR
    UECK INS BASICPROGRAMM                <048>
260 REM                                   <068>
270 REM -------- USR-VEKTOR AUF $828 RICHT
    EN -----                              <139>
280 REM                                   <088>
290 POKE 785,60 :REM LSB DES USR-VEKTORS  <065>
300 POKE 786,3  :REM MSB DESSELBEN        <061>
310 REM                                   <118>
320 REM -------- EINGABEN UND USR-AUFRUF -
    ---------                             <145>
330 REM                                   <138>
340 PRINT CHR$(147):INPUT"ZAHL EINGEBEN";A <014>
350 B=USR(A):REM B IST NUR EIN DUMMY      <189>
360 REM                                   <168>
370 REM ----- PROGR.DIREKTMODUS : MONITORA
    UFRUF -----                           <212>
380 PRINT CHR$(147)CHR$(17)               <235>
390 PRINT"SYS49152"CHR$(17)CHR$(17)CHR$(17
    )CHR$(17)                             <088>
400 PRINT" M 6000 6001"CHR$(17):REM HIER L
    IEGT DIE ZAHL IM MFLPT-FORMAT         <146>
410 PRINT" M 6010 6011"CHR$(17):REM UND HI
    ER IM FLPT-FORMAT                     <213>
420 PRINT" X"CHR$(17)                     <046>
430 PRINT"RUN490"                         <196>
440 PRINT CHR$(19);                       <035>
450 POKE 631,13                           <002>
460 POKE 198,1:END                        <045>
470 REM ----------------------------------
    -----------                           <195>
480 REM                                   <032>
490 PRINT:PRINT"AB $6000 MFLPT-FORMAT"    <076>
500 PRINT"AB $6010 FLPT-FORMAT"           <167>
510 PRINT:INPUT"WEITERE ZAHLEN (J/N)";A$  <179>
520 IF A$="J" THEN 340                    <106>
530 POKE 785,72:POKE 786,178:REM USR-VEKTO
    R AUF NORMALWERT                      <037>
540 END                                   <034>
```

Listing 1. Berechnung von FLPT- und MFLPT-Format für den C 64

```
10 REM ************************************
   ***********
20 REM *
          *
30 REM *   PROGRAMM ZUM UMWANDELN VON ZAHLEN
    IN DIE   *
40 REM *   C128-FORMATE    MFLPT   (AB $6000)
          *
50 REM *                    FLPT   (AB $6010)
          *
60 REM *
          *
70 REM *       HEIMO PONNATH  HAMBURG  1986
          *
80 REM *
          *
90 REM ************************************
   ***********
100 REM
110 FOR I=0 TO 17: REM EINLESEN DES MASCHINE
    NPROGRAMMES
120 READ D$: POKE DEC("1600")+I,DEC(D$)
130 NEXT I
140 REM -------- DAS MASCHINENPROGRAMM -----
    -------
150 DATA A2,00   : REM LDX #$00     ;LSB ZIE
    LADRESSE
160 DATA A0,60   : REM LDY #$60     ;MSB
    -"-
170 DATA 20,03,8C: REM JSR $8C03    ;FAC ->
    (X/Y)
180 DATA A2,06   : REM LDX #$06     ;ZAEHLER
     EINRICHTEN
190 DATA B5,62   : REM LDA $62,X    ;FAC AUS
    LESEN
200 DATA 9D,0F,60: REM STA $600F,X  ;UND UEB
    ERTRAGEN
210 DATA CA      : REM DEX          ;ZAEHLER
     -1
220 DATA D0,F8   : REM BNE $1609    ;WEITER
    BIS FAC UEBERTRAGEN IST
230 DATA 60      : REM RTS          ;ZURUECK
    INS BASICPROGRAMM
240 REM
250 REM -------- USR-VEKTOR AUF $1600 RICHTE
    N -----
260 REM
270 POKE DEC("1219"),0 : REM LSB DES USR-VEK
    TORS
280 POKE DEC("121A"),22: REM MSB DESSELBEN
290 BANK 15: REM SICHERHEITSHALBER
300 REM
310 REM -------- EINGABEN UND USR-AUFRUF ---
    -------
320 REM
330 PRINT CHR$(147): INPUT "ZAHL EINGEBEN";A
340 B=USR(A): REM B IST NUR EIN DUMMY
350 REM
360 REM ----- PROGR.DIREKTMODUS : MONITORAUF
    RUF -----
370 PRINT CHR$(147) CHR$(17)
380 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(1
    7) CHR$(17)
390 PRINT "M 06000 06001" CHR$(17): REM HIER
     LIEGT DIE ZAHL IM MFLPT-FORMAT
400 PRINT "M 06010 06011" CHR$(17): REM UND
    HIER IM FLPT-FORMAT
410 PRINT "X" CHR$(17)
420 PRINT "RUN480"
430 PRINT CHR$(19);
440 BANK 0: POKE 842,13: POKE 843,13: POKE 8
    44,13: POKE 845,13: POKE 846,13
450 POKE 208,5: END
460 REM ----------------------------------
    ---------
470 REM
480 PRINT : PRINT "AB $6000 MFLPT-FORMAT"
490 PRINT "AB $6010 FLPT-FORMAT"
500 PRINT : INPUT "WEITERE ZAHLEN (J/N)";A$
510 IF A$="J" THEN 330
520 POKE DEC("1219"),40: POKE DEC("121A"),12
    5: REM USR-VEKTOR AUF NORMALWERT
530 END
```

Listing 2. Und dasselbe für den C 128