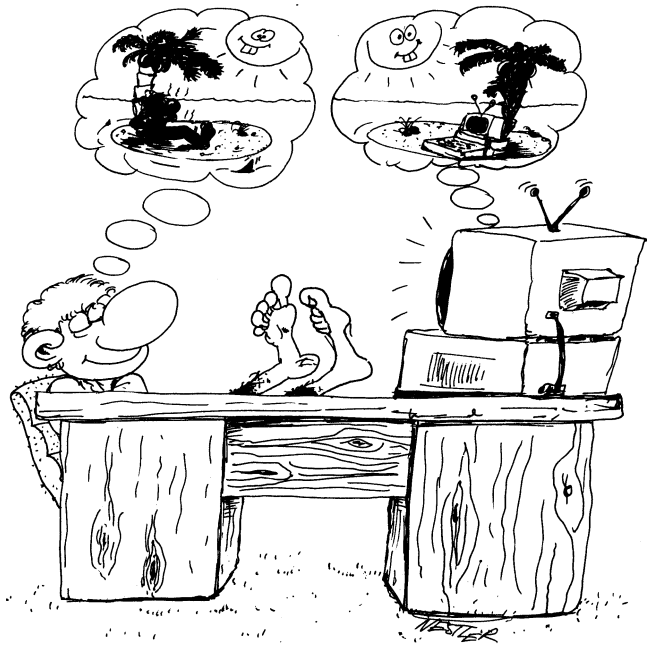
```
■ (ZP,X) : indiziert indirekt
Der Inhalt des X-Registers wird zum zweistelligen,
hexadezimalen Operanden addiert und ergibt eine Adresse
in der Zeropage (Speicherbereich $0000 bis $00FF).
Deren Inhalt und der Inhalt der darauffolgenden Adresse
ergibt in der Form Lo-Byte/Hi-Byte die Arbeitsadresse.

Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDX #$0E
LDA ($12,X)
Der Inhalt der Zeropage-Adressen $0020 ($000E + $0012)
und $0021 ergibt die Arbeits-Adresse $C000. Deren
Inhalt wird in den Akku geladen.

■ (ZP),Y : indirekt indiziert
Der zweistellige, hexdezimale Operand ergibt eine
Adresse in der Zeropage (Speicherbereich $0000 bis
$00FF). Deren Inhalt und der Inhalt der darauffolgenden
Speicherstelle ergibt in der Form Lo-Byte/Hi-Byte eine
Adresse, zu der der Inhalt des Y-Registers addiert
wird. Das Ergebnis ist die Arbeitsadresse.

Beispiel:
Adresse $20 hat den Inhalt $00
Adresse $21 hat den Inhalt $C0
LDY #$10
LDA ($20),Y
Der Inhalt der Adresse $C010 ($C000 + $0010) wird in
den Akku geladen.
```

Tabelle 3. Diese Abkürzungen werden in den Tabellen 1 und 2 verwendet (Schluß)



ROM-Routinen in eigenen Programmen

Das Rad ist schon erfunden! Ähnlich verhält es sich mit verschiedenen Routinen, die ein Assembler-Programmierer immer wieder benötigt. Aber warum soll man sich die Arbeit des Programmierens machen, wenn das Betriebssystem viele ständig benötigte Routinen schon enthält und man nur noch zu wissen braucht, ab welcher Adresse sie stehen?

Angenommen, Sie möchten in Assembler einige komplexe Dinge programmieren wie beispielsweise eine neue mathematische Funktion (wie wäre es mit dem Kotangens) und diese auf dem Bildschirm ausgeben. Das ist eine große Aufgabe, zu der zunächst einmal die Übernahme des Arguments in das Maschinenprogramm, dann einige Fließkomma-Rechenoperationen und schließlich die Ausgabe auf dem Bildschirm geschrieben werden müßten, wenn da nicht schon fast alles an verborgener Stelle als fertige Programm-Module im Computer vorhanden wäre!

Sowohl im unteren (von $A000 bis $BFFF) als auch im oberen ROM-Bereich (von $E000 bis $FFFF) liegt die Firmware fest verschachtelt vor. Der untere ROM-Abschnitt wird manchmal auch Basic-Interpreter, der obere ROM-Bereich Betriebssystem genannt, wobei diese Einteilung aber den Kern der Sache nicht genau trifft, denn Interpreter, Editor und Betriebssystem führen ein gemischtes Dasein quer durch alle genannten ROM-Bereiche hindurch.

Mindestens fünf Informationen braucht ein Assembler-Programmierer, wenn er das breite Programmangebot des ROMs nutzen möchte:

1. Einsprungadresse
2. Format der Eingabeparameter
3. Adressen der Eingabeparameter
4. Adressen der Ausgabeparameter
5. Format der Ausgabeparameter

Nicht alle Routinen, die man benutzen kann, erfordern alle fünf Informationen, manche weniger, einige auch mehr und schließlich gibt es noch Programmroutinen, die noch den Aufruf einer oder sogar mehrerer anderer Routinen nötig machen.

In der beigefügten Tabelle sind – nach Anwendungen sortiert – die wichtigsten Firmware-Möglichkeiten mit den erforderlichen Ein- und Ausgabeparametern aufgeführt. Das sind natürlich beileibe nicht alle. Die Auswahl erfolgte subjektiv! Es sind einfach diejenigen, die mir bislang am häufigsten untergekommen sind. Außerdem wurde auf die Kernel-Routinen verzichtet: Man findet diese sehr gut dokumentiert bereits in einer Reihe von Büchern und im Assembler-Kurs.

Die Tabelle nennt den Label-Namen, die Einsprungadresse und gibt eine Kurzbeschreibung der Funktion. Das Ein- und auch das Ausgabeformat ist ebenso angegeben wie auch die Adressen, an denen diese Parameter übergeben werden. Die verwendeten Bezeichnungen halten sich eng an die im Assembler-Kurs kennengelernten. Sie sind allgemein üblich:

FAC	Fließkomma-Akku 1
ARG	Fließkomma-Akku 2
A	Akkumulator
X,Y	X-, Y-Register
A/Y	2-Byte-Angabe im Format LSB/MSB im Akku/Y-Register
FLPT	Fließkommazahl im Normalformat
MFLPT	gepacktes Fließkommaformat

Damit das alles nicht so trocken abläuft, soll noch ein kleines Beispiel vorgestellt werden! Die oben schon erwähnte Kotangens-Funktion wird in einem Maschinenprogramm erzeugt, das durch USR anzuspringen ist. In Bild 1 finden Sie ein Flußdiagramm zu dem Programm, welches hier als Hypra-Ass-Listing abgebildet ist (Listing 1). Ein kurzes Testprogramm liefert Listing 2.

Der Einsprung mittels USR bietet den Vorteil, daß der Übergabewert gleich im FLPT-Format im FAC »landet«. Es ist aber sinnvoll, den Übergabeparameter mittels der MOVMF-Routine zu »retten«, weil durch die Kosinus-Funktion der FAC verändert wird. Wenn auch das Ergebnis der Kosinus-Funktion mittels MOVMF beiseite gelegt wurde, holen wir durch MOVFM den Anfangswert wieder in den FAC und bilden mittels SIN den Sinus davon. Schließlich teilen wir den im Speicher stehenden Kosinuswert durch den im FAC befindlichen Sinuswert (unter Verwendung von FDIV). Das Ergebnis ist der Kotangens:

COT X = (COS X/SIN X)

Dieser Wert befindet sich nun im FAC und wird mit dem RTS an das Basic-Programm zurückgeliefert. Im Testprogramm weisen wir ihm dann die Variable E zu.

Dieses kurze Beispiel soll Ihnen den Mund wässrig machen. Sehr viel detaillierter werden die ROM-Routinen im Kurs »Von Basic zu Assembler« im 64'er behandelt werden.

(Heino Ponnath/hm)

Literatur:
1. Kassera/Kassera, Programmieren in Maschinensprache, München 1985: Markt&Technik Verlag, MT 830
2. West, C64 Computerhandbuch, München 1984, Te-wi
3. Babel/Krause/Dripke, Das Interface Age Systemhandbuch zum C 64, München 1983: Interface Age Verlag
4. Ponnath, C 64 Wunderland der Grafik, München 1985: Markt&Technik Verlag MT 756.

```
hypra-ass  assemblerlisting:

                10   -            .li 1,4,7
                20   -            .ba $6000
;
;einsprung mittels usr
;zuvor usr-vektor einstellen!
;
                160  -            .eq cos=$e264
                165  -            .eq movfm=$bba2
                170  -            .eq movmf=$bbd4
                180  -            .eq sin=$e26b
                190  -            .eq fdiv=$bb0f
                200  -            .eq wert=$7000
                205  -            .eq wert1=$7010
;
6000 a210    :212  -start         ldx #<(wert1)
6002 a070    :214  -              ldy #>(wert1)
6004 20d4bb  :216  -              jsr movmf
6007 2064e2  :220  -              jsr cos
600a a200    :230  -              ldx #<(wert)
600c a070    :240  -              ldy #>(wert)
600e 20d4bb  :250  -              jsr movmf
6011 a910    :252  -              lda #<(wert1)
6013 a070    :254  -              ldy #>(wert1)
6015 20a2bb  :256  -              jsr movfm
6018 206be2  :260  -              jsr sin
601b a900    :270  -              lda #<(wert)
601d a070    :280  -              ldy #>(wert)
601f 200fbb  :290  -              jsr fdiv
6022 60      :300  -              rts
;
                320  -            .sy 1,4,7

symbols in alphabetical order:

cos                = $e264
fdiv               = $bb0f
movfm              = $bba2
movmf              = $bbd4
sin                = $e26b
start              = $6000
wert               = $7000
wert1              = $7010

end of assembly  0:25.9
base = $6000  last byte at $6022
```

Listing 1. Hypra-Ass-Listing der Kotangens-Funktion

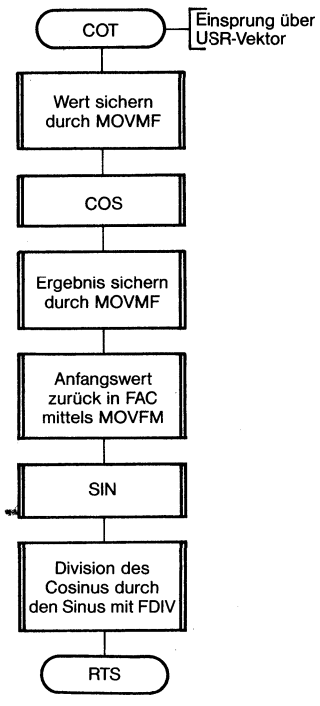


Bild 1. Flußdiagramm einer Kotangens-Funktion

```
10 REM***TEST FUER COTANGENS***
20 POKE785,0:POKE786,96:REM USR-VEKTOR
30 INPUT"WINKEL";W:W=W*π/180:REM AUF BOGENMASS
40 E=USR(W):REM AUFRUF DES PROGRAMMES
50 PRINTW,E:REM ERGEBNIS IN E
60 END
READY.
```

Listing 2. Test der Kotangens-Funktion

1. Routinen, die die Kooperation von Basic und Assembler erleichtern:

Label	Adresse	Funktion	Eingabe Format	Eingabe Adresse	Ausgabe Format	Ausgabe Adresse
CHRGET	0073	Holt nächstes Byte	1 Byte	Basic-Text	1 Byte	A
CHRGOT	0073	Holt aktuelles Byte	1 Byte	Basic-Text	1 Byte	A
READY	A474	Erzeugt READY-Status	-	-	-	-
LINGET	A96B	Holt Integerwert (0-63999)	ASCII-Zahl	Basic-Text	2-Byte Integer	14/15
FRMNUM	AD8A	Holt beliebigen numerischen Ausdruck	Basic-Ausdruck	Basic-Text	FLPT	FAC
FRMEVL	AD9E	Holt beliebigen Ausdruck	Basic-Ausdruck	Basic-Text	a) bei Fließkomma: FLPT b) bei Integer: FLPT c) bei String: Zeiger auf Descriptor	FAC FAC FAC+3/FAC+4

Diese Routine setzt außerdem eine Reihe von Flaggen:
VALTYP($0D) 0=Zahl FF=String
INTFLAG ($0E) 0=Fließkomma 80=integer

		War Ausdruck einfache Variable, dann zeigt VARNAM ($45/6) das 1. Byte des Variablen-Namens				
CHKCLS	AEF7	Prüft auf ») «	ASCII	Basic-Text	-	-
CHKOPN	AEFA	Prüft auf » («	ASCII	Basic-Text	-	-
CHKCOM	AEFD	Prüft auf »,«	ASCII	Basic-Text	-	-
SYNCHR	AEFF	Prüft auf Zeichen im Akkumulator	ASCII	Basic-Text A	-	-
		Diese 4 Routinen überlesen das Zeichen, wenn vorhanden. Wenn nicht vorhanden, folgt SYNTAX ERROR				
ISVAR	AF28	Sucht Variablenwert	Name + Kennung	$45/46	a) Zahl: FLPT b) String: Descriptor-FAC+3	FAC
ORDVAR	B0E7	Sucht Variablennamen	Name+ Kennung	$45/46	Adresse	$47/48
GTBYTC	B79B	Holt Zahl (0-255)	ASCII	Basic-Text	1 Byte	X
GETNUM	B7EB	Liest 2 Integerzahlen (Trennung durch Komma) 1. Zahl: 0 bis 65535 2. Zahl: 0 bis 255	ASCII	Basic-Text	2Byte-Int. 1Byte-Int.	$14/15 X
COMBYT	E200	Prüft auf »,« und holt folgende Zahl	ASCII	Basic-Text	1 Byte	X

2. Routinen, die Verschiebungen im Speicher durchführen:

BLTUC	A3BF	Verschiebt Blöcke	Adressen: Quelle Start Ende+1 Ziel	$5F/60 $5A/5B		
			Ende+1	$58/59	-	-
PUTINT	A9C4	Schiebt FAC als Integer in Variable	FLPT Adresse	FAC $49/50	2Byte-Integer	angegebene Variable
PTFLPT	A9D6	Schiebt FAC in Variable	FLPT Adresse	FAC $49/50	MFLPT	angegebene Variable
GETSPT	AA2C	Schiebt String-descriptor in Variable	Zeiger Adresse	FAC+3 $49/50	Descriptor	angegebene Variable
STRVAL	B7B5	Zahlenstring in FAC einlesen	ASCII Länge	ab $22 A	FLPT	FAC
CONUPK	BA8C	Lädt ARG aus Speicher	MFLPT	A/Y	FLPT	ARG
MOVFM	BBA2	Lädt FAC aus Speicher	MFLPT	A/Y	FLPT	ARG
MOVMF	BBD4	Schiebt FAC in Speicher	FLPT Adresse	FAC X/Y	MFLPT	angegebener Speicher
MOVFA	BBFC	ARG in FAC kopieren	FLPT	ARG	FLPT	FAC
MOVAF	BC0C	FAC in ARG kopieren	FLPT	FAC	FLPT	ARG
ACTOFC	BC3C	Akku in FAC schieben	1Byte	A	FLPT	FAC

3. Routinen zur Arithmetik:

ASCADD	AA27	Addiert ASCII-Ziffer zu FAC	ASCII Ziffer	A	FLPT	FAC
OROP	AFE6	FAC=(FAC)OR(ARG)	FLPT	FAC,ARG	FLPT	FAC
ANDOP	AFE9	FAC=(FAC)AND(ARG)	FLPT 0	FAC,ARG Y	FLPT	FAC
FACINX	B1AA	FAC wird als Integer in A/Y abgelegt	FLPT	FAC	2Byte-Integer	A/Y
UMULT	B357	16-Bit-Multiplikation	2-Byte-Integer Zahl1 Zahl2	$28/29 $71/72	2Byte-Integer	X/Y
CIVAYF	B391	Integer (-32768 bis 32767) in FAC	2Byte-Integer	A/Y	FLPT	FAC
SGNFT	B3A2	Integer (0 bis 255) in FAC	1Byte	y	FLPT	FAC
GETADR	B7F7	Wandelt FAC zu Integer (0-65535)	FLPT	FAC	2Byte-Integer	Y/A +$14/15
FADDH	B849	FAC = FAC + 0,5	FLPT	FAC	FLPT	FAC
FSUB	B850	FAC=Speicherzahl -FAC	MFLPT FLPT	Zeiger A/Y FAC	FLPT	FAC
FSUBT	B853	FAC = ARG - FAC	FLPT	ARG,FAC	FLPT	FAC
FADD	B867	FAC=Speicherzahl +FAC	MFLPT MFLPT	Zeiger A/Y FAC	FLPT	FAC
FADDT	B86A	FAC = ARG + FAC	FLPT	ARG,FAC	FLPT	FAC
COMPLT	B947	Erzeugt Zweier-komplement von FAC	FLPT	FAC	FLPT	FAC
LOG	B9EA	FAC = ln(FAC)	FLPT	FAC	FLPT	FAC
FMULT	BA28	FAC=Speicherwert*FAC	MFLPT FLPT	Zeiger A/Y FAC	FLPT	FAC
FMULTT	BA30	FAC = ARG * FAC	FLPT	ARG,FAC	FLPT	FAC
MUL10	BAE2	FAC = 10 * FAC	FLPT	FAC	FLPT	FAC
DIV10	BAFE	FAC = FAC/10	FLPT	FAC	FLPT	FAC
FDIVF	BB07	FAC=ARG/Speicher-zahl	MFLPT FLPT	Zeiger A/Y ARG	FLPT	FAC
FDIV	BB0F	FAC=Speicherzahl/FAC	MFLPT FLPT	Zeiger A/Y FAC	FLPT	FAC
FDIVT	BB14	FAC = ARG/FAC	FLPT	FAC,ARG	FLPT	FAC
SIGN	BC28	Ermittelt Vorzeichen von FAC	FLPT	FAC	1Byte 1 ← + 0 ← 0 FF ← -	A
ABS	BC58	FAC = ABS(FAC)	FLPT	FAC	FLPT	FAC
FCOMP	BC5B	Vergleicht FAC mit Speicherzahl	MFLPT FLPT	Zeiger A/Y FAC	1Byte: 1: FAC > Speicher 0: FAC = Speicher FF: FAC < Speicher	A
INT	BCCC	FAC = INT(FAC)	FLPT	FAC	FLPT	FAC
AADD	BD7E	Addiert A zu FAC	FLPT 1Byte	FAC A	FLPT	FAC
SQR	BF71	FAC = SQR(FAC)	FLPT	FAC	FLPT	FAC
MPOT	BF78	FAC=Speicherwert ↑ FAC	FLPT MFLPT	FAC Zeiger A/Y	FLPT	FAC
FPWRT	BF7B	FAC = ARG ↑ FAC	FLPT	ARG,FAC	FLPT	FAC
NEGOP	BFB4	FAC = -FAC	FLPT	FAC	FLPT	FAC
EXP	BFED	FAC = e↑FAC	FLPT	FAC	FLPT	FAC
POLYX	E059	Polynomberechnung FAC=a0+a1x+a2x²+... Zeiger weist auf Start der Konstantentabelle. 1. Byte = Polynomgrad Weitere Bytes sind die Koeffizienten des Polynoms in der Reihenfolge an,.....,a0 im MFLPT-Format.	Adresse	Zeiger A/Y	FLPT	FAC
COS	E264	FAC = COS(FAC)	FLPT	FAC	FLPT	FAC
SIN	E26B	FAC = SIN(FAC)	FLPT	FAC	FLPT	FAC
TAN	E2B4	FAC = TAN(FAC)	FLPT	FAC	FLPT	FAC
ATN	E30E	FAC = ATN(FAC)	FLPT	FAC	FLPT	FAC

4. Auswahl von Ein-/Ausgabe-Routinen:

ERROR	A437	Fehlermeldung ausgeben und READY	Fehler-nummer	X	ASCII	Bildschirm
LIST	A69C	Listet Basic-Programm	-	-	-	-
NUMDON	AABC	Druckt FAC auf Bildschirm aus	FLPT	FAC	ASCII	Bildschirm
STROUT	AB1E	Gibt String auf Bildschirm aus. Ende=0	Adresse	Zeiger A/Y	ASCII	Bildschirm
SYNERR	AF08	Ausgabe SYNTAX ERROR	-	-	ASCII	Bildschirm
OVERR	B97E	Ausgabe OVERFLOW ERR.	-	-	ASCII	Bildschirm
LINPRT	BDCD	Druckt Integerzahl (0 bis 65535) aus.	2Byte-Integer	X/A	ASCII	Bildschirm
FACOUT	BDD7	Druckt FAC auf Bildschirm aus	FLPT	FAC	ASCII	Bildschirm
FOUT	BDDD	FAC wird zu ASCII-String (Ende=0). Kann direkt mit STROUT ausgegeben werden.	FLPT	FAC	ASCII (Ende=0) Startadr. A/Y	ab $100
SAVET	E156	Save	Parameter aus Basic-Text			
VERFYT	E165	Verify	Parameter aus Basic-Text			
LOADT	E168	Load	Parameter aus Basic-Text			
SLPARA	E1D4	Holt Parameter für Save, Verify, Load aus dem Basic-Text				
PLOTK	E50A	Setzt Cursorposition	Zeile Spalte	X Y		
HOME	E566	Cursor in Home-Position				
PLOTR	E56C	Setzt Cursor-Position	Zeile Spalte	$D6 $D3		
GETKBC	E5B4	Holt Zeichen aus Tastaturpuffer	-	-	1Byte	A
PRT	E716	Gibt Zeichen in A auf Bildschirm aus	1Byte	A	ASCII	Bildschirm
CLRLN	E9FF	Löscht xte Bildschirmzeile	Zeilen-nummer	X	-	-

Tabelle der ROM-Routinen