

PETER'S ASSEMBLIERECKE

Floating-Point Arithmetik

Diesmal packt die Assemblerecke ein Thema an, über das in amerikanischen und erst recht in deutschen Zeitschriften noch nicht viel zu lesen war. Es geht um die Routinen zum Rechnen mit Fließkommazahlen, die im "Mathe-ROM" fest in jedem Atari vorhanden sind.

Das ROM der Atari XL/XE-Computer ist in drei vollkommene getrennte Programmpakete unterteilt: In das Betriebssystem, auf das wir in der Assemblerecke schon mehrfach zurückgegriffen haben, weiterhin das allseits bekannte Atari-Basic und schließlich noch das Mathematik-Paket. Letzteres wird in der Fachliteratur meist als Mathe-ROM oder FP-ROM bezeichnet, wobei FP für Floating Point oder Fließkomma-rechnung steht.

Dieses FP-ROM soll in der Assemblerecke genauer unter die Lupe genommen werden. Nicht so versierte Leser der Assemblerecke werden jetzt vielleicht fragen, wozu man so etwas überhaupt benötigt. Nun ja, die Assemblerbefehle des 6502-Prozessors enthalten nur Befehle zum Addieren und Subtrahieren von acht Bit breiten Binärzahlen, die einen Zahlenbereich von 0 bis 255 erlauben. Für viele Anwendungen ist dieser Bereich aber einfach zu klein. Stellen Sie sich nur vor, Sie wollen Ihre Haushaltskasse per Programm verwalten, dann dürften nicht mehr als DM 2.55 drin sein. Oder wenn Sie Ihren Atari gar bei wissenschaftlichen Berechnungen verwenden wollten, dann wären Sie mit diesem Zahlenbereich hoffnungslos verloren. Man hilft sich daher und baut aus den elementaren Rechenbefehlen der Maschinensprache komplexere Rechenoperationen auf, die mit einem größeren Zahlenbereich arbeiten können. Praktisch ist

zu diesem Zweck die Darstellung mit Gleitkomma, die Sie bestimmt vom Taschenrechner her kennen.

Das Mathe-ROM ist nichts weiter als eine recht umfangreiche Sammlung von Unterprogrammen, die mit FP-Zahlen rechnen und diese auch in andere Darstellungen wandeln können. Die Leser der Assemblerecke haben bereits eine Anwendung solcher Umwandlungen kennengelernt: In der Ausgabe 11/85 wurden zwei Routinen des FP-ROMs verwendet, um Zahlen so umzuwandeln, daß sie auf dem Bildschirm ausgegeben werden können.

Der "prominenteste" Benutzer des FP-ROMs ist das Atari-Basic, das dort alle Rechenoperationen mit Hilfe der Unterprogrammen des Mathe-Paketes durchführt. Da die Routinen (mit wenigen Ausnahmen) auch ohne Basic zur Verfügung stehen, können sie auch von anderen Programmiersprachen benutzt werden. Pascal und LOGO benutzen das FP-ROM, bei Forth und ACTION! kann das Mathe-Paket über die Erweiterungen angesprochen werden. Natürlich können wir die Fähigkeiten der Mathematik-Routinen auch in Assembler benutzen, dazu soll auch das Programmbeispiel nähere Erläuterungen geben.

Damit Ihre Erwartungen nicht zu hoch gespannt sind, muß gleich eines vorausgeschickt werden: Die Routinen des FP-ROMs sind durch die Bank ziemlich langsam. Das ist übrigens auch einer der Gründe, warum das Atari-Basic nicht eben ein Renner ist. Einige neuere Basic-Versionen für den Atari, darunter das Basic XE, bringen daher ihre eigenen, schnelleren FP-Routinen gleich selbst mit.

Benutzung

Das FP-ROM liegt im Speicherbereich von \$D800 bis \$DFFF. Es belegt also 2 kByte ROM-Speicherplatz. Natürlich brauchen die Routinen auch etwas Speicherplatz im RAM, denn dort müssen Zwischenergebnisse und diverse Arbeitsregister Platz finden. Zu diesem Zweck sind gleich zwei Bereiche reserviert: In der Zero-Page ab Adresse \$D4 bis \$FF und in der Page 5 ab Adresse \$580 bis \$5FF. Eine besondere Bedeutung kommt den Adressen in der Zero-Page zu, denn dort befinden sich die beiden "Software Akkumulatoren" für die FP-Rechenoperationen. Diese beiden Akkus sind in Wirklichkeit je sechs aufeinander folgende Speicherzellen, die jeweils eine FP-Zahl aufnehmen können. Gerechnet wird dann, indem einer oder beide FP-Akkus geladen und die gewünschte Rechenoperation aufgerufen wird.

Die beiden Akkus heißen FR0 (ab \$D4) und FR1 (ab \$E0). Weiterhin gibt es noch die beiden Pointer FLPTR und INBUFF, deren Funktion später noch erklärt wird. Die einzelnen Funktionen des Mathematik-Paketes werden über festgelegte Einsprungsadressen durch einen Unterprogramm-sprung mit JSR aufgerufen. Diese Adressen sind tatsächlich in allen 8-Bit-Atari-Computern gleich, d. h., ein Programm, das FP benutzt, läuft auf einem alten 400 genauso wie auf einem neuen 130XE. Im folgenden Abschnitt stehen nur die Namen der verwendeten Routinen, die genauen Einsprungsadressen entnehmen Sie bitte dem Definitionsteil von Listing 1.

Zuerst drängt sich die Frage auf, wie die FP-Akkus mit einem bestimmten Wert geladen werden können. Dazu gibt es zwei Routinen, die eine FP-Zahl ab einer beliebigen Adresse in einen FP-Akku laden. Die Adresse wird im X-(LSB) und im Y-Register (MSB) übergeben. Die Unterprogramme haben die Namen FLD0R (steht für Floating Load 0 via Register) und FLD1R und wirken auf Akku FR0 und FR1 je nach der Zahl im Namen. Ab der angegebe-

nen Adresse wird eine gültige, 6 Byte lange FP-Zahl vermutet, die Adresse selbst wird im Zeiger FLPTR hinterlegt. Möchte man die gleiche Zahl nochmal laden, so kann man sich der Einsprungsadressen FLD0P und FLD1P (Floating Load via Pointer) bedienen. Wie man eine FP-Zahl richtig im Speicher ablegt, können Sie im Kästen nachlesen.

Ganz analog läßt sich der Inhalt des FP-Akkus null mit FST0R und FST0P (via Pointer) im Speicher ablegen. Weiterhin gibt es das Unterprogramm FMOVE, mit dem der Inhalt von FR0 nach FR1 kopiert werden kann. Die Routine ZFR0 dient zum Löschen von FR0.

Rechnen

Für die nachfolgende Rechenoperation müssen beide FP-Akkus zuvor mit den gewünschten Werten vorbesetzt werden, anschließend wird das jeweilige Unterprogramm aufgerufen. Alle Grundrechenarten sind vorhanden: Multiplikation mit FMUL, Division mit FDIV, Addition und Subtraktion von Gleitkommazahlen mit FADD und FSUB. Bei der Division wird FR0 durch FR1 geteilt und das Ergebnis in FR0 abgelegt, gleiches gilt für die Subtraktion.

Darüber hinaus gibt es noch einige Routinen, die nur ein Argument benötigen. Es handelt sich um den natürlichen Logarithmus (LOG), den Logarithmus zur Basis 10 (LOG10) und die entsprechenden Exponentialfunktionen zur Basis e (EXP) und zur Basis 10 (EXP10). Für alle Rechenroutinen gilt: Ist nach dem Aufruf das Carry-Flag gesetzt, so ist etwas schief gelaufen, d. h. der Zahlenbereich ist überschritten worden.

Zahlenumwandlungen

Neben den reinen Rechenfunktionen gibt es noch eine Reihe von Hilfsprogrammen zur Umwandlung von verschiedenen Formen der Zahlendarstellungen. Interessant für den Assemblerprogrammierer sind die Funktionen FPI und IFP, die eine Umwandlung von 16-Bit Integers in Gleitkommazahlen und zurück erlauben. Für IFP (Integer nach FP) wird die Integerzahl in die Speicherzel-

len FR0 (\$D4, LSB) und FR0+1 (\$D5, MSB) eingetragen, nach dem Aufruf von IFP entsteht die FP-Darstellung FP-Akku 0. Genau umgekehrt arbeitet die Routine FPI, d. h., FP-Akku 0 wird vorbesetzt und danach kann die Integerzahl aus FR0 und FR0+1 entnommen werden.

Will man die Zahlen am Bildschirm ausgeben oder von der Tastatur einlesen, so braucht man eine Darstellung der Zahl im ATASCII-Code. Auch hier hilft das FP-ROM mit zwei Unterprogrammen. FASC wandelt eine Gleitkommazahl in FR0 in die entsprechende Darstellung im ATASCII-Code um. Das Ergebnis der Umwandlung ist im sogenannten Line-Buffer (LBUFF, ab \$580) zu finden, und als weitere Hilfe wird der Zeiger INBUFF (\$F3, \$F4) auf das erste Zeichen gerichtet. Das letzte Zeichen des erzeugten ATASCII-Strings ist markiert, indem sein höchstwertiges Bit auf eins gesetzt ist. Eine Routine zum Ausdruck eines solchen Strings ist in der Assemblerecke von CK 11/85 (Ausgabe von Zahlen) zu finden.

Umgekehrt ist eine Umrechnung vom ATASCII-Code in die FP-Darstellung mit der Routine AFP möglich. Die Adresse des Strings muß in INBUFF eingetragen werden. Zusätzlich kann noch ein Index CIX (in \$F2) vorgegeben werden. Das ist recht nützlich, wenn Sie z. B. wissen, daß in einer Eingabe die Zahl immer ab der 5. Position steht. In diesem Fall müßten Sie INBUFF auf den Anfang des Strings und CIX auf 5 setzen. Gewandelt wird so lange, bis ein Zeichen angetroffen wird, das nicht mehr Teil der Zahl sein kann. Das Ergebnis wird in FR0 abgelegt. Ist das Carry-Flag nach AFP gesetzt, wurde kein wertbares Zeichen gefunden.

Trigonometrie

Ihnen ist vielleicht aufgefallen, daß trigonometrische Funktionen wie $\sin(x)$ und $\cos(x)$ bisher noch nicht erwähnt wurden. Der Grund ist naheliegend: Sie sind im FP-ROM nicht enthalten. Wahrscheinlich aus Platzgründen wurden diese Funktionen ins

Atari-Basic mit aufgenommen und sind daher nicht in allen Fällen erreichbar. An dieser Stelle gleich eine Warnung: Wenn Sie in irgend einer Publikation für den Sinus eine Adresse von z. B. \$BE05 finden, dann Vorsicht! Je nach Revision des Basics kann diese Adresse anders sein. Bleibt na-

türlich die Frage, wie man denn nun zu einem Sinus kommen soll. Dazu dient die letzte der FP-Routinen PLYEVL, mit der Potenzreihen berechnet werden können. Wenn Sie mit der Mathematik auf Kriegsfuß stehen, empfehle ich Ihnen, den nächsten Abschnitt zu überlesen.

Mathematik pur

Eine Potenzreihe ist eine Summe von unendlich vielen Produkten, die jeweils aus einem Faktor und einer Potenz der unabhängigen Variablen gebildet werden. Speziell der Sinus kann mit folgender Reihe berechnet werden:

Gleitkomma unter der Lupe

Die Darstellung einer Zahl mit Gleitkomma ist Ihnen sicherlich vom Taschenrechner her geläufig. Dieses Zahlenformat bietet den Vorteil, daß sowohl sehr große als auch sehr kleine Zahlen mit gleichbleibender Genauigkeit dargestellt werden können. Fließkommazahlen bestehen aus zwei Teilen: dem Exponenten und der Mantisse.

Um von einer normal dargestellten Zahl in das Gleitkommaformat zu kommen, verschiebt man das Komma an eine zuvor festgelegte Stelle und merkt sich dabei, wie oft und in welcher Richtung die Verschiebung vorgenommen wurde. Diese Anzahl der Verschiebungen notiert man anschließend im Exponenten. Eine solche Verschiebung entspricht genau einer Multiplikation bzw. Division mit 10, daher spricht man von einem Exponenten zur Basis 10.

Beim Atari wurde eine etwas andere Definition benutzt, der Exponent ist hier zum Beispiel 100. Das bedeutet konkret, daß das Komma immer zwei Stellen verschoben werden muß, wenn der Exponent um 1 erhöht wird. Dazu ein Beispiel (das E hinter den Zahlen steht für "100 hoch"):

$$12345.678 = 123.45678 \text{ E}+1 = 1.2345678 \text{ E}+2$$

Die letzte Darstellung ist die im Atari verwendete Normalform. Das Komma wird so verschoben, daß entweder eine oder zwei Ziffern noch links davon stehen. Im obigen Fall bleibt nur die Eins vor dem Komma, denn bei nochmaliger Verschiebung würde der Teil vor dem Komma zu Null. Ist die Zahl kleiner als eins, so wird der Exponent entsprechend negativ:

$$0.00123456 = 0.123456 \text{ E}-1 = 12.3456 \text{ E}-2$$

Um positive und auch negative Exponenten in einem einzigen Byte unterzubringen, bedient man sich eines Tricks. Man addiert zum Exponenten immer 64 und hat somit einen Zahlenbereich von -64 bis +63. Das würde reichen, um Zahlen mit 126 Nullen vor dem Komma bzw. 128 Nullen nach dem Komma darzustellen. Tatsächlich benutzt werden allerdings nur Zahlen mit den Absolutbeträgen von 10 hoch 98 bis 10 hoch minus 98.

Wie Sie sich leicht überlegen können, kommt der Exponent in diesem Format mit nur 7 Bit aus und man kann das höchstwertige Bit des Exponentenbytes noch heranziehen, um das Vorzeichen der ganzen Zahl (nicht zu verwechseln mit dem Vorzeichen des Exponenten!) zu verschlüsseln. Null bedeutet hier positiv, eins heißt, daß die Zahl negativ ist.

Die Mantisse, damit bezeichnet man die Zifferninformation der Zahl, wird beim Atari im BCD-Code dargestellt. BCD steht für "binär codierte Dezimalziffer" und ist an sich eine recht einfache Sache. Immer zwei dezimale Ziffern werden in einem Byte zusammenfaßt und als hexadezimale Zahl betrachtet:

53 dez. in BCD: \$53 (hex.)

Insgesamt sind fünf Byte für die Mantisse einer Gleitkommazahl vorgesehen, das heißt, daß bis zu 10 Ziffern eingetragen werden können. Die beiden obigen Beispiele sehen dann so aus:

+ 12345.678 wird zu \$42 \$01 \$23 \$45 \$67 \$80

- 0.00123456 wird zu \$BE \$12 \$34 \$56 \$00 \$00

Das jeweils erste Byte ist der Exponent, im ersten Fall $64+2=66$ (\$42) und bei der negativen Zahl $64-2+128=190$ (\$BE). Die Ziffern können dank des BCD-Codes direkt als Hex-Ziffern übernommen werden. Einige zusätzliche Beispiele finden Sie auch im Listing 1. Interessant ist vielleicht noch, daß durch die Darstellung mit Exponenten zur Basis 100 die Genauigkeit der Fließkommazahlen im Atari zwischen neun und zehn Dezimalstellen schwankt. Sie beträgt maximal neun Ziffern, wenn vor dem Komma nur eine Stelle besetzt ist (z. B. die obige positive Zahl) und maximal zehn Ziffern, wenn zwei Stellen vor dem Komma bleiben (vgl. mit der obigen negativen Zahl). Eine Sonderstellung nimmt die Darstellung der Null ein, hier sind alle sechs Byte Null.

Peter Finzel

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

Klar, daß man auch einem Computer nicht unendlich viele Reihenglieder zumuten kann. Man beschränkt sich daher je nach gewünschter Genauigkeit auf drei bis sechs Glieder, je mehr man rechnet, um so genauer wird es, da die Reihe konvergiert. Mit PLYEVL kann man solche Reihen leicht berechnen. Man übergibt im Akku die Anzahl der Glieder und im X- (LSB) und Y-Register einen Zeiger, der auf den Anfang einer Tabelle mit den Koeffizienten (beginnend mit dem letzten $F(n)!$) deutet. Wie man den Sinus und den Cosinus mit PLYEVL berechnet, können Sie aus den Unterprogrammen SIN und COS in Listing 1 ersehen. Einschränkend muß man zu diesen Routinen sagen, daß sie nur den ersten Quadranten berechnen, sonst wird eine Fallunterscheidung nötig. Außerdem werden nur drei Glieder berechnet, wodurch die Genauigkeit etwas vermindert wird (Basic berechnet sechs!).

Beispiel Kreise

Um Ihnen das Rechnen mit dem FP-ROM etwas näher zu bringen, finden Sie in Listing 1 ein längeres Beispiel eines Assemblerprogrammes, das Gleitkommaarithmetik benutzt. Es kann vom Basic mit $X = \text{USR}(28672, \text{XM}, \text{YM}, \text{RAD})$ aufgerufen werden und zeichnet einen Kreis mit dem Radius RAD um den Mittelpunkt XM, YM. Vorher muß ein GRAPHICS-Befehl erfolgen und die Zeichenfarbe mit COLOR eingestellt werden. Ein Demo finden Sie in Listing 2, das auch das Assemblerprogramm als Basic-Loader enthält. Erwarten Sie aber bitte nicht, daß die Kreise blitzschnell gezeichnet werden, denn wie gesagt, ist das FP-ROM nicht gerade von der schnellen Truppe. Aber ein biß-

chen flotter als in Basic geht's schon. Da das Programm hauptsächlich Demo-Charakter hat, ist es nicht gegen Überschriften durch Basic geschützt. Sie sollten deshalb darauf achten, daß Ihr Basic-Programm nicht zu lange wird (bis \$7000). Sicherlich wird Ihnen dieses Programm den Umgang mit den FP-Routinen näherbringen. Zum Beispiel wird gezeigt, wie man zwei FP-Zahlen vergleicht: Einfach voneinander subtrahieren und das Bit 7 der Speicherzelle FRO mit BPL oder BMI prüfen. Einfach aber wirksam!

Das Programm wurde mit dem ATMAS-II Makroassembler geschrieben und benutzt die Möglichkeit des Offsets. Mit ORG \$7000, \$A800 wird das Programm so assembliert, daß es ab Adresse \$7000 lauffähig ist. Es wird während der Assemblierung aber im freien Speicherbereich ab \$A800 abgelegt. Im Monitor kann es dann mit der INTO-Option so abgespeichert werden, daß es beim erneuten Laden (z. B. mit DOS) an die richtige Adresse gelangt. Auf diese Weise kann man mit ATMAS-II Programme für jeden beliebigen Speicherbereich schreiben. Mit einigem Geschick ist es sogar möglich, Programme ins RAM parallel zum Betriebssystem zu bekommen.

Das Demoprogramm berechnet ein Achtel eines Kreisbogens, der achtmal gespiegelt wird, um einen vollständigen Kreis zu bekommen. Die Koordinaten werden über $X = R * \cos(a)$ und $Y = R * \sin(a)$ berechnet, wobei der Winkel a von Null auf 45 Grad ($\pi/4$ im Bogenmaß) erhöht wird. Als Schrittweite für den Winkel wird 1/R (im Bogenmaß) genommen, damit ergibt sich ein geschlossener Kreis ohne Löcher.

Peter Finzel

Listing 1

```
*****
* KREISE ZEICHNEN MIT FLOATING POINT
*
* Aufruf: X=USR(Adr,Xm,Ym,Radius)
*
* Assembler: ATMAS-II
*
* Peter Finzel 1986
*****
* Einsprungsadressen des FP-ROMs:
*
AFP EQU $DB00 ATASCII nach FP
```

```
FASC EQU $DBE6 FP nach ATASCII
IFP EQU $D9AA Integer nach FP
FPI EQU $D9D2 FP zu Integer
FSUB EQU $DA60 Subtraktion
FADD EQU $DA66 Addition
FMUL EQU $DADB Multiplikation
FDIV EQU $DB28 Division
FLDOR EQU $DB89 FRO via Register laden
FLDOP EQU $DB8D FRO via Pointer laden
FLDIR EQU $DB98 FRI via Register laden
FLDIR EQU $DB9C FRI via Pointer laden
FSTOR EQU $DDA7 FRO via Register speichern
FSTOP EQU $DDAB FRO via Pointer speichern
FMOVE EQU $DDB6 FRO nach FRI kopieren
PLYEVL EQU $DD40 Potenzreihe berechnen
EXP EQU $DDC0 Exponentialfunktion e^X
EXP10 EQU $DDCC Exponentialfkt. 10^X
LOG EQU $DECD natürlicher Logarithmus
LOG10 EQU $DED1 Zehner-Logarithmus
ZFRO EQU $DA44 FRO löschen
```

* Arbeitsregister der FP-Routinen

```
FRO EQU $D4 FP-Akku 0
FLPTR EQU $FC Zeiger (Pointer)
DEGFLG EQU $FB 0: Bogenmass, 6: Grad
```

* ATARI-Systemkonstante

```
ICCOM EQU $342 IOCB-Konstante
ICBLL EQU $348
ICBLH EQU $349
CIOV EQU $E456 CIO-Einsprung
CPBIN EQU $0B Put Binary
ROWCRS EQU $54 Cursorzeile
COLCRS EQU $55 Cursorspalte
COLOR EQU $C8 COLOR-Wert von BASIC
```

* PLOT-ROUTINE FUER KREIS

```
* ORG $7000,$A800 laeuft ab $7000
*
FPKREIS PLA Argumente
LDA #0 Bogenmass
STA DEGFLG einschalten
PLA
STA FRO+1 'X' vom Stack
PLA
STA FRO
JSR IFP in FP wandeln
LDX #XMFP und speichern
LDY #YMFP/256
JSR FSTOR via Register
PLA
STA FRO+1 'Y' vom Stack
PLA
STA FRO
JSR IFP in FP
LDX #YMFP
LDY #YMFP/256
JSR FSTOR
PLA
STA FRO+1 'R' vom Stack
PLA
STA FRO
JSR IFP
LDX #RADFP
LDY #RADFP/256
JSR FSTOR
```

* 1/R fuer Schrittweite des Winkels

```
*
JSR FMOVE
LDX #EINS
LDY #EINS/256 Eins in FP-
JSR FLDOR Darstellung
JSR FDIV in FRO
LDX #STEPFP dividieren
LDY #STEPFP/256 Schrittweite
JSR FSTOR merken
```

* Schleifenzaehler auf Null

```
*
JSR ZFRO FRO löschen
LDX #GRADFP
LDY #GRADFP/256
JSR FSTOR
```

* Schleife

```
*
SCHLEIF LDX #GRADFP zuerst den
LDY #GRADFP/256 Sinus berechnen
JSR FLDOR
JSR SIN
LDX #RADFP mal Radius
LDY #RADFP/256
JSR FLD1R Y=sin(x)*R
JSR FMUL
LDX #YFP
LDY #YFP/256
JSR FSTOR als Y speichern
```

* Y=cos(x)*R

```
*
LDX #GRADFP dann Cosinus
LDY #GRADFP/256 berechnen
JSR FLDOR
JSR COS
LDX #RADFP mit Radius
LDY #RADFP/256 multiplizieren
JSR FLD1R
JSR FMUL
LDX #XFP
LDY #XFP/256
JSR FSTOR
```

* JSR PLOT4 4 Punkte plotten

* jetzt die X und Y-Werte vertauschen
* damit wird der Kreis aus insgesamt
* 8 Segmenten zu je 45 Grad gebildet.

* LDX #5

```

XYTSCH LDA XFP,X
PHA PHA
LDA YFP,X
STA XFP,X
PLA
STA YFP,X
DEX
BPL XYTSCH

JSR PLOT4 nochmal 4 Punkte

*
* Step-Wert zum momentanen Winkel
* addieren: GRAD=GRAD+STEP
*

LDA #GRADFP
LDY #GRADFP/256
JSR FLDOR Winkel laden
LDX #STEPFP
LDY #STEPFP/256 Schrittweite
JSR FLDIR laden
JSR FADD addieren
LDX #GRADFP und wieder
LDY #GRADFP/256 speichern
JSR FSTOR
JSR FMOVE

*
* 1st GRAD schon groesser als PI/4?
*

LDA #PIVIER Winkel steht
LDY #PIVIER/256 noch in FR1
JSR FLDOR PI/4 in FRO
JSR FSUB PI/4 - GRAD
LDA FRO Ergebnis < 0?
BMI ENDE Ja, Ende!
JMP SCHLEIF weiter -->

ENDE RTS zu BASIC

*****
* Unterpgm. plottet 4 Punkte des Kreises
*****
PLOT4 LDA #0
JSR SEGLOT
LDA #40
JSR SEGLOT
LDA #80
JSR SEGLOT
LDA #C0
JSR SEGLOT
RTS

*****
* Unterprogramm zum Plotten eines Kreispunktes
*
* Flag bestimmt, ob XM+X oder XM-X bzw/
* YH+Y oder YM-Y
* berechnet wird.

SEGLOT STA FLAG
LDX #XFP
LDY #XFP/256 X und XM laden
JSR FLDIR
LDX #YMFP
LDY #YMFP/256
JSR FLDOR
BIT FLAG
BVS SUB1
JSR FADD
JMP SEG1
JSR FSUB
JSR FFI

LDA FRO
STA COLCRS
LDA FRO+1
STA COLCRS+1

*
* Y-Koordinate bestimmen
*

LDX #YFP
LDY #YFP/256 Y und YM laden
JSR FLDIR
LDX #YMFP
LDY #YMFP/256
JSR FLDOR
BIT FLAG
BMI SUB2
JSR FADD
JMP SEG2
JSR FSUB
JSR FFI

LDA FRO+1 >255??
BNE KEINPLT nicht erlaubt!
LDA FRO in Cursorzeile
STA ROWCRS eintragen

*
* es folgt der PLOT-Befehl
*

LDX #60 IOCB #6
LDA #C0 Put Binary
LDA #0 ein Punkt
STA ICBLL,X
STA ICBLL,X
LDA COLOR aktuelle Farbe
JSR CIOV
KEINPLT RTS

*
*****
* Berechnen des SINUS nach der Funktionenreihe
* sin(x)=x*(1 - x^2/3! + x^4/5! - ...)
* Abbruch nach dem 3. Glied
*****

SIN LDX #FPHILF
LDY #FPHILF/256

JSR FSTOR Argument merken
JSR FMOVE nach FR1
JSR FMUL Argument quadrieren
LDA #3 3 Glieder
LDX #SINKOEF Koeffizienten-
LDY #SINKOEF/256 tabelle
JSR PLYEVL Potenzreihe P(x*x)

LDX #FPHILF Argument laden
LDY #FPHILF/256
JSR FLDIR
JSR FMUL und: x*P(x*x)
RTS

*
* Die Koeffiziententabelle der Sinusreihe
* als Brueche: 1/120, -1/6, 1
*
SINKOEF DFB $3E,$83,$33,$33,$33,$33
DFB $BF,$16,$66,$66,$66,$67
EINS DFB $40,$01,$00,$00,$00,$00

*****
* Berechnung des Cosinus nach der Funktionen-
* reihe cos(x)=1 - x^2/2! + x^4/4! ...
* Abbruch nach dem dritten Glied
*****

COS JSR FMOVE Argument
JSR FMUL quadrieren
LDA #3 3 Koeffizienten
LDX #COSKOEK Tabelle der
LDY #COSKOEK/256 Koeffizienten
JSR PLYEVL Potenzreihe
RTS fertig!

*
* Koeffiziententabelle des COS:
* als Brueche: 1/24, 1/2, 1
*
COSKOEK DFB $3F,$04,$66,$66,$66,$67
DFB $BF,$50,$00,$00,$00,$00
DFB $40,$01,$00,$00,$00,$00

*
* 1/4 * PI (45 Grad) in FP-Darstellung
*
PIVIER DFB $3F,$78,$53,$98,$16,$34
*
FLAG DFB 0 Merker f. verschiedene Zwecke
*
RADFP ORG **6 Speicherplatz fuer
GRADFP ORG **6 FP-Variable
STEPFP ORG **6
YMFP ORG **6
XFP ORG **6
YFP ORG **6
FPHILF ORG **6
*

```

Listing 2

```

100 REM *****
110 REM * LISTING 2: Basic-Loader *
120 REM * DEMO ZUM KREISPROGRAMM *
130 REM * P. FINZEL 1986 *
150 REM *****
200 GOSUB 30000
210 KREIS=28672
300 GRAPHICS 8+16:COLOR 1
310 FOR R=5 TO 50 STEP 5
320 X=USR(KREIS,159,R,R)
330 X=USR(KREIS,159,191-R,R)
340 X=USR(KREIS,159-R,95,R)
350 X=USR(KREIS,159+R,95,R)
360 NEXT R
400 GOTO 400
30000 REM * KREIS-ROUTINE
30010 S=0:RESTORE 30100
30020 FOR A=28672 TO 29093:READ D:POKE
  A,D:S=S+D:NEXT A
30030 IF S<>52984 THEN ? "DATEN-FEHLER
  !":STOP
30090 RETURN
30100 DATA 104,169,0,133,251,104,133,2
  13,104,133,212,32,170,217,162
30110 DATA 184,160,113,32,167,221,104,
  133,213,104,133,212,32,170,217

```

30120 DATA 162,190,160,113,32,167,221,
104,133,213,104,133,212,32,170
30130 DATA 217,162,166,160,113,32,167,
221,32,182,221,162,119,160,113
30140 DATA 32,137,221,32,40,219,162,17
8,160,113,32,167,221,32,68,218
30150 DATA 162,172,160,113,32,167,221,
162,172,160,113,32,137,221,32
30160 DATA 74,113,162,166,160,113,32,1
52,221,32,219,218,162,202,160
30170 DATA 113,32,167,221,162,172,160,
113,32,137,221,32,125,113,162
30180 DATA 166,160,113,32,152,221,32,2
19,218,162,196,160,113,32,167
30190 DATA 221,32,207,112,162,5,189,19
6,113,72,189,202,113,157,196
30200 DATA 113,104,157,202,113,202,16,
239,32,207,112,162,172,160,113
30210 DATA 32,137,221,162,178,160,113,
32,152,221,32,102,218,162,172
30220 DATA 160,113,32,167,221,32,182,2
21,162,159,160,113,32,137,221
30230 DATA 32,96,218,165,212,48,3,76,8
3,112,96,169,0,32,228,112,169
30240 DATA 64,32,228,112,169,128,32,22
8,112,169,192,32,228,112,96
30250 DATA 141,165,113,162,196,160,113
,32,152,221,162,184,160,113

```
30260 DATA 32,137,221,44,165,113,112,6
,32,102,218,76,3,113,32,96,218
30270 DATA 32,210,217,165,212,133,85,1
65,213,133,86,162,202,160,113
30280 DATA 32,152,221,162,190,160,113,
32,137,221,44,165,113,48,6,32
30290 DATA 102,218,76,42,113,32,96,218
,32,210,217,165,213,208,24,165
30300 DATA 212,133,84,162,96,169,11,15
7,66,3,169,0,157,72,3,157,73
30310 DATA 3,165,200,32,86,228,96,162,
208,160,113,32,167,221,32,182
30320 DATA 221,32,219,218,169,3,162,10
7,160,113,32,64,221,162,208
30330 DATA 160,113,32,152,221,32,219,2
18,96,62,131,51,51,51,51,191
30340 DATA 22,102,102,102,103,64,1,0,0
,0,0,32,182,221,32,219,218,169
30350 DATA 3,162,141,160,113,32,64,221
,96,63,4,102,102,102,103,191
30360 DATA 80,0,0,0,0,64,1,0,0,0,0,63,
120,83,152,22,52,0
```