

Entwicklerdokumentation

zur maschinennahen Programmierung

TGI Praktikum SS03

Gruppe 29

Manuel Mayr
Schulstraße 4
85356 Attaching
mayrm@in.tum.de

Alexander Gafriller
Schulstraße 4
85356 Attaching
gafriller@in.tum.de

20. Juli 2003

Inhaltsverzeichnis

1	Einführung	2
1.1	Idee des Algorithmus	2
1.2	Taylor-Reihen	2
1.2.1	Entwicklung der Taylorreihe	3
1.3	Interpolationstabelle	3
1.4	Das Zusammenspiel	3
1.4.1	Prinzipieller Ablauf	4
2	Implementation	5
2.1	Architektur	5
2.2	Variablen und Konstanten	5
2.2.1	Konstanten	6
2.2.2	Initialisierte Variablen	6
2.2.3	Uninitialisierte Variablen	7
2.3	Die Funktionen	8
2.3.1	fast_sin	8
2.3.2	init_table	10
2.3.3	taylor_sin	12
2.3.4	fmod	15
2.3.5	floor_mode	17
2.3.6	default_mode	18

Kapitel 1

Einführung

1.1 Idee des Algorithmus

Der Algorithmus der die `fast_sin`-Funktion definiert, versucht mittels seiner Basiskomponenten eine elegante Lösung zwischen Effizienz und Präzision zu finden. Er bedient sich auf dem Gebiet der Analysis der Taylor-Reihen, die zwar in der Berechnung langsam sind aber die Sinus-Funktion sehr genau approximieren. Um die Hürde der Geschwindigkeit zu überwinden ziehen wir ein ziemlich naives aber äußerst effektives Konzept in Erwägung. Beim ersten Aufruf der Funktion, berechnen wir mittels Taylor-Reihen einige "Stützstellen" der Sinus-Funktion. Diese speichern wir in einer Tabelle ab. Bei weiteren Aufrufen interpolieren wir dann zwischen den Werten, die in der Tabelle gespeichert wurden und versuchen so ein möglichst schnelles und genaues Ergebnis zu erzielen.

1.2 Taylor-Reihen

Die Stützstellen der Tabelle werden der Genauigkeit wegen, mit der bewährten Methode der Taylor-Reihe berechnet.

Kurz zum Hintergrund:

Ist eine Funktion im Intervall $[a, b]$ beliebig oft stetig differenzierbar, so lässt sich diese Funktion als Potenzreihe darstellen.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

$x_0 \in [a, b]$ nennen wir dabei den Entwicklungspunkt und $\frac{f^{(n)}(x_0)}{n!}$ die Taylorkoeffizienten.

1.2.1 Entwicklung der Taylorreihe

Um die Funktion $y = \sin(x)$ zu errechnen wählen wir $x_0 = 0$. Daraus ergibt sich folgende Lösung:

$$(i) f^{(0)}(x) = \sin(x) \implies f^{(0)}(0) = \sin(0) = 0$$

$$(ii) f^{(1)}(x) = \cos(x) \implies f^{(1)}(0) = \cos(0) = 1$$

$$(iii) f^{(2)}(x) = -\sin(x) \implies f^{(2)}(0) = -\sin(0) = 0$$

$$(iv) f^{(3)}(x) = -\cos(x) \implies f^{(3)}(0) = -\cos(0) = -1$$

$$(v) f^{(4)}(x) = \sin(x) \implies f^{(4)}(0) = \sin(0) = 0$$

Wir sehen an der obigen Berechnungsfolge, dass sich mit weiteren Ableitungen die Zeilen (i) - (iv) weiter fortsetzen. Wir sehen auch, dass die geraden Koeffizienten der Taylorreihe wegfallen und so können wir im nächsten Schritt folgende Formel entwickeln:

$$\sin(x) = \sum_{n=1}^{\infty} \frac{x^{(2 \cdot n - 1)}}{(2 \cdot n - 1)!} (-1)^{(n+1)}$$

Die Reihe sieht dann wie folgt aus:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} \dots$$

Durch diese Reihe lässt sich der Sinus beliebig genau approximieren.

1.3 Interpolationstabelle

Die Tabelle die wir für diesen zweck im Speicher reserviert haben kann 512 Stützstellen fassen. Wir hatten hier zwar mehrere Werte vorgesehen, die im Speicher stehen sollten, aber dies scheiterte an implementierungsspezifischen Problemen. Da die Sinus-Funktion mit der Frequenz $\omega = 2 \cdot \pi$ periodisch ist, enthält die Tabelle natürlich nur Werte die zum Eingabeintervall von $[0, 2 \cdot \pi]$ passen.

1.4 Das Zusammenspiel

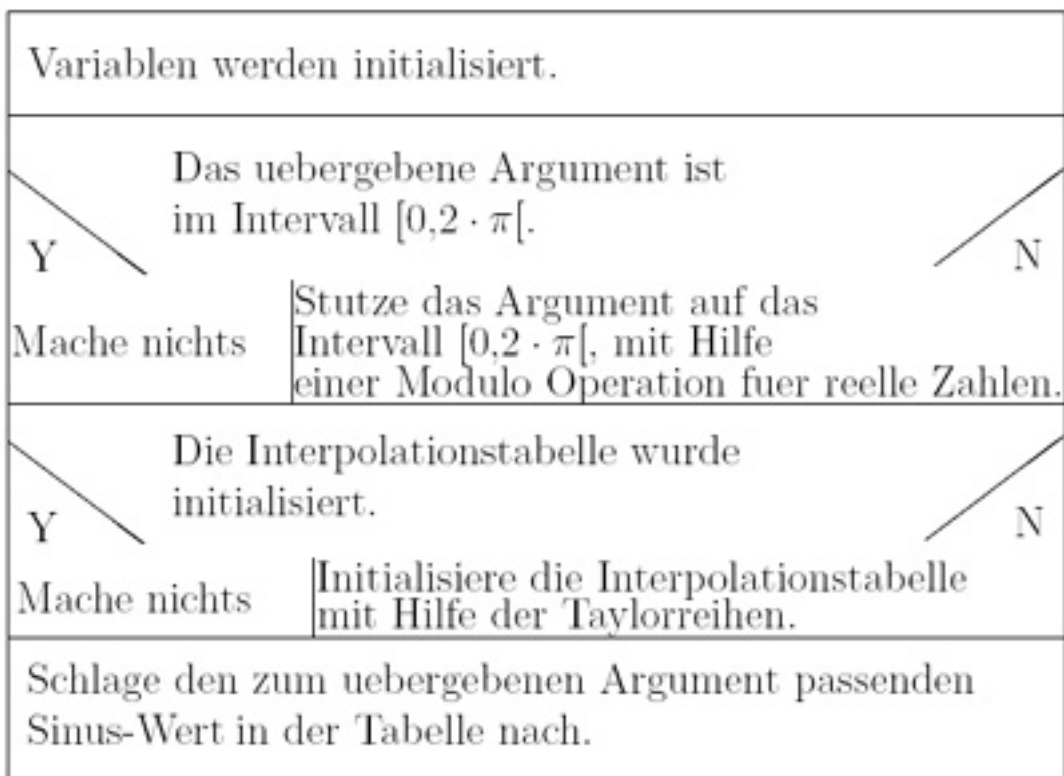
Beim ersten Aufruf der `fast_sin`-Funktion werden alle 512 Stützstellen berechnet und in der Tabelle gespeichert. Dies kann unter Umständen einige

Zeit dauern, weshalb der Befehl erst mit folgenden Aufrufen an Effizienz gewinnt. Bei diesen entfallen die zeitaufwändigen Berechnungen und der Algorithmus bezieht sich nur mehr auf die bereits initialisierte Tabelle. Der Eingabewert wird zweckmäßiger Weise wegen der Periodizität der Sinus-Funktion auf das Intervall $[0, 2 \cdot \pi]$ reduziert. In der Tabelle wird dann nach dem zur Eingabe passenden Funktionswert gesucht.

1.4.1 Prinzipieller Ablauf

Der Prinzipielle Ablauf des Programms soll im Folgenden durch ein Nassi-Schneidermann-Diagramm illustriert werden.

Ablauf — Berechnung des Sinus mit Interpolationstabelle



Kapitel 2

Implementation

Im folgenden Kapitel wird auf die Implementation des `fast_sin`-Befehls eingegangen. Alle Codefragmente die im Folgenden verwendet werden stammen direkt aus der Datei `sinus.asm` in der die Funktion `fast_sin` definiert wird.

2.1 Architektur

Der Befehl wurde für einen zum Intel 80386 kompatiblen PC programmiert. Für die korrekte Anwendung und Funktionsweise ist das Vorhandensein einer zur 80387 Fließkommazahleinheit (FPU) von Intel oder eines entsprechenden Emulators unbedingt vonnöten.

2.2 Variablen und Konstanten

Bei der Implementierung mussten mehrere Konstanten und Variablen definiert werden, deren Funktionsweise im Folgenden erläutert werden. Zuerst folgt die Abbildung des ersten Codefragments auf das sukzessive eingegangen wird:

Ausschnitt aus `sinus.asm`

```
%define ENTITY_SIZE 8           ; Laenge einer Einheit in der Tabelle
%define TABLE_SIZE 512        ; Groesse der Tabelle

segment .data
    minus_one dt -1.           ; enthaelt den Wert -1
    lookup_size dd 512         ; enthaelt die Groesse der Tabelle
```

```
segment .bss
    controlWordReg resw 1      ; (wird fuer die Berechnung benoetigt
                                ; Kontroll-Wort-Register
    faculty rest 1            ; Speichervariable fuer den aktuellen
                                ; Wert der Fakultaet
    faculty_count rest 1      ; enthaelt den Wert fuer den die
                                ; Fakultaet zu berechnen ist
    square_x rest 1           ; Speichervariable fuer x*x
    taylor_val rest 1         ; Speichervariable fuer den aktuellen
                                ; Wert der Reihe
    plus_minus_one rest 1     ; Speichervariable fuer den alternierenden
                                ; Teil des aktuellen Gliedes
    counter resd 1
    lookup_table resq TABLE_SIZE ; die Lookup Tabelle zwischen deren
                                ; Werten interpoliert wird
    index resd 1
```

Ausschnitt aus `sinus.asm`

2.2.1 Konstanten

Hier werden ausschließlich im Programm definierte Konstanten erläutert.

ENTITY_SIZE Diese Konstante definiert die Größe einer Zelle der Interpolationstabelle in Bytes. Der Wert der abgelegt werden kann entspricht einer Fließkommazahl. Der in der Programmiersprache C dazu äquivalente Datentyp ist `double`.

TABLE_SIZE Diese Konstante definiert die Größe der Tabelle, also die Anzahl der Zellen bzw. Werte, welche in ihr abgelegt werden. Der Speicher den die Tabelle belegt hat also eine Größe von $512 \cdot 8 = 4096$ Bytes.

2.2.2 Initialisierte Variablen

In diesem Bereich werden jene Variablen erläutert welche im `data`-Segment des Codes stehen und somit einen Initialisierungswert enthalten.

`minus_one` Um die alternierende Reihe zu erzeugen, welche die Sinus-Funktion approximiert, muss sehr oft mit -1 multipliziert werden um die negativen Reihenglieder zu erzeugen. -1 kann so einfach auf den Stack der FPU untergebracht und für Berechnungen zur Rate gezogen werden.

`lookup_size` enthält noch einmal die Größe der Tabelle, wie die Konstante `TABLE_SIZE`. Allerdings kann mit einer Konstante nicht gerechnet werden. `lookup_size` dient vorwiegend zur Indexberechnung.

2.2.3 Uninitialisierte Variablen

In diesem Bereich werden jene Variablen erläutert welche im `bss`-Segment des Codes stehen und somit keinen Initialisierungswert enthalten, sondern nur eine Position im Speicher zugewiesen bekommen, in dem sie Werte ablegen können.

`controlWordReg` Diese Variable wird später in der Prozedur `floor_mode` dazu verwendet um das Kontroll-Word-Register in jenen Modus zu versetzen, der die FPU beim Runden einer Zahl dazu veranlässt die Nachkommastellen abzuschneiden.

`faculty_count` enthält den aktuellen Wert, für den die Fakultät berechnet werden soll.

`faculty` wird im späteren Algorithmus verwendet um den aktuellen Wert der Fakultät, die ja essenziell für die Erzeugung der Taylor-Reihe ist, zu speichern.

`square_x` birgt in sich den Wert x^2 , der bei der Erzeugung der Reihe einer äußerst wichtige Rolle spielt, da der Wert im Zähler der Reihenglieder immer mit x^2 multipliziert werden muss, um das darauffolgende Glied zu berechnen.

`taylor_val` Im Algorithmus wird er später den aktuellen Wert der Reihe enthalten und somit schlussendlich einen ziemlich genauen Wert für den Sinus an der Stelle x .

`plus_minus_one` kann den Wert -1 oder $+1$ annehmen, je nachdem welches Vorzeichen das aktuelle Reihenglied hat.

`counter` ist eine Zählvariable, die im Code sehr flexibel zum Einsatz kommt.

`lookup_table` Diese Variable ist von besonderer Relevanz, da sie die Abstraktion der Interpolations-Tabelle `real` im Speicher darstellt. Sie stellt einen durchgehenden Block im Speicher dar und ist eine zentrale Komponente für den Algorithmus.

`index` Wird nur zur Indexberechnung verwendet und hat sonst keine andere Funktionalität.

2.3 Die Funktionen

In diesem Abschnitt wird vorwiegend auf die Algorithmen und Funktionen eingegangen. Dabei wird so vorgegangen, dass zuerst die Funktionen erklärt werden, welche rudimentäre Aufgaben erfüllen und erst zum Schluss werden jene Funktionen erklärt, die dann ins Detail der Implementation gehen.

2.3.1 `fast_sin`

Die Funktion `fast_sin` ist jene Funktion, die dann in einem C-Programm vom Endbenutzer aufgerufen wird. Sie vereinigt die Teilroutinen, die später noch erläutert werden zu einem Algorithmus, der schlussendlich den Sinus eines Argumentes $x \in \mathfrak{R}$ berechnet.

Ausschnitt aus `sinus.asm`

```

1:_fast_sin
2:enter 0,0                ; nimmt Initialisierungen fuer den
3:                        ; Aufruf durch das Rahmenprogramm vor
4:                        ; Speicherung des Base-Pointers und
5:                        ; der Ruecksprungadresse
6:
7: mov eax, [ebp+8]        ; holt die Adresse des Argumentes,
8:                        ; welche als erstes Element uebergeben wurde
9:                        ; vom Stack
10:    fld QWORD [eax]     ; Laedt den uebergebenen Wert
11:
12: cmp DWORD [counter], TABLE_SIZE-1 ; ist der Wert des counters
13:                        ; gleich der Groesse der Tabelle
14:                        ; so wurde die Tabelle schon

```

```

15:                                     ; initialisiert
16: jae beginning                       ; Initialisierung wird uebersprungen
17: call init_table                     ; Tabelle wird initialisiert
18:beginning:
19: call fmod                           ; Begrenzung des uebergebenen Wertes
20:                                     ; auf [0,2*PI[
21: fild DWORD [lookup_size]
22: fldpi
23: fld st0
24:     faddp st1, st0
25: fdivp st1, st0
26: fmulp st1, st0                       ; Berechnung des Index in der der
27:                                     ; zum Wert zugehoerige Wert steht
28: call floor_mode                       ; Nachkommastellen abschneiden
29: frndint                              ; st0 runden
30: call default_mode                    ; Normalen Round-Precision Mode
31:                                     ; einstellen
32:
33: fistp DWORD [index]
34: mov eax, ENTITY_SIZE
35:     mul DWORD [index]                 ; Index wird mit der Groesse einer
36:                                     ; Einheit multipliziert
37: fld QWORD [lookup_table+eax]         ; speichern des Wertes der Tabelle
38:                                     ; in st0
39:
40: leave                                ; Austritt aus der Funktion (fuers Rahmenprogramm)
41: ret

```

Ausschnitt aus `sinus.asm`

7 - 10 Hier holt die Funktion lediglich das Argument, das der Funktion als erstes uebergeben wurde vom Stack. Dieses Argument ist natuerlich jener x -Wert ($x \in \mathbb{R}$) fuer den der Sinus berechnet werden muss. Dieser Wert wird fuer die weitere Berechnung in den Stack der FPU geladen.

12 - 17 Dieses Codefragment dient zum Überprüfen des Vorhandenseins der Interpolationstabelle. Hat der `counter` bereits die Größe der Tabelle angenommen, dann ist diese bereits initialisiert worden und die folgende Routine

zum Initialisieren der Tabelle kann übersprungen werden. Ist dies nicht der Fall, dann muss die Tabelle initialisiert werden.

19 Der Parameter $x \in \mathfrak{R}$ wird auf das Intervall $[0, 2 \cdot \pi[$ eingegrenzt.

21 - 30 Im Folgenden werden wir den Index berechnen. Dazu gehen wir so vor wie bereits in der Spezifikation erklärt. Das Codestück berechnet also die Formel $index = \frac{lookup_size}{2 \cdot \pi} \cdot x$. Den Nachkommateil müssen wir allerdings noch abschneiden, da eine Tabelle nur einen Index im Bereich der natürlichen Zahlen hat. Dies passiert in den Zeilen 28 - 30.

33 - 34 Der soeben berechnete Index wird in einer Variable gespeichert. Um die physikalische Position der Zelle im Speicher zu finden muss er nun mit der Größe einer Zelle multipliziert werden.

37 Zur Adresse der Tabelle wird zum Schluss noch die eben berechnete Position hinzuaddiert und das Ergebnis in das Register `st0` (top of stack) der FPU geschrieben, wo laut C-Calling-Convention der Rückgabewert der Funktion stehen soll.

40 - 41 Austritt aus der Funktion.

2.3.2 `init_table`

Die Aufgabe dieser Routine ist es die Interpolationstabelle mit entsprechenden Sinus-Werten zu füllen. Dazu wird die `taylor_sin`-Funktion verwendet um die Stützstellen so genau wie möglich approximieren zu können.

Ausschnitt aus `sinus.asm`

```

1: init_table
2: mov DWORD [counter], 0           ; counter initialisieren
3: push ebx                         ; ebx sichern, darf nicht
4:                                     ; geändert werden
5:     mov ebx, lookup_table         ; ebx mit der Adresse der
6:                                     ; Lookup-Table initialisieren
7:
8:     repeat:

```

```
9:          fldpi                ; pi auf st0 speichern
10:         fld st0
11:         faddp st1, st0        ; 2*pi berechnen
12:         fild DWORD [lookup_size] ; Groesse der Tabelle auf st0
13:                                     ; schreiben
14:
15:         fdivp st1, st0        ; 2*pi / lookup_size berechnen
16:         fild DWORD [counter]
17:         fmulp st1, st0        ; (2*pi / lookup_size) * counter
18:                                     ; berechnen
19:
20:         wait
21:         call taylor_sin        ; Sinus-Wert berechnen
22:         fstp QWORD [ebx]       ; in der Tabelle abspeichern
23:
24:         cmp DWORD [counter], TABLE_SIZE-1 ; ueberpruefen ob schon genug
25:                                               ; Werte in der Tabelle sind
26:
27:         jae until              ; wenn die Tabelle gefuehlt
28:                                               ; ist ans Ende spingen
29:
30:         wait
31:         add ebx, ENTITY_SIZE    ; neue Adresse der Tabelle fuer
32:                                     ; naechsten Wert berechnen
33:         add DWORD [counter], 1 ; counter inkrementieren
34:
35:         jmp repeat              ; an den Anfang der Iteration springen
36:     until:
37:         pop ebx                 ; ebx wird unveraendert zurueckgelassen
38:         ret
```

Ausschnitt aus `sinus.asm`

2 - 5 Hier werden einfach nur einige Werte initialisiert. Die Zählvariable `counter`, welche die Einträge in der Tabelle abzählt, wird auf den Anfangswert 0 gesetzt. Das Register `ebx` soll die Anfangsadresse der Tabelle im Speicher enthalten.

8 Hier beginnt eine Schleife, sodass alle Zellen der Tabelle gefüllt werden können.

9 - 17 Hier wird das aktuelle Argument berechnet, welches der `taylor_sin`-Funktion übergeben wird und für das der aktuelle Sinus-Wert berechnet werden soll, der in der Tabelle eingetragen wird.

Dazu verwenden wir jene Funktion, die bereits in der Spezifikation für diesen Zweck verwendet wurde. Mit $x = \frac{2 \cdot \pi}{lookup_size} \cdot counter$ berechnen wir also das Argument dessen Sinus-Wert an der Position `counter` in der Tabelle eingetragen wird.

20 - 23 Die Funktion `taylor_sin` berechnet den Sinus von x , der dann sogleich in der Tabelle abgespeichert wird.

24 - 35 Der Vergleich dient zur Überprüfung ob die Tabelle schon voll ist. Sollte das der Fall sein, wird die Routine abgebrochen. Im anderen Fall wird die Zählvariable `counter` inkrementiert und die Adresse für die nächste Zelle berechnet

37 - 38 Das Register `ebx` darf laut C-Calling-Convention nicht verändert werden und deshalb wird der ursprüngliche Wert wieder zurückgeschrieben. Der darauf folgende Befehl verlässt die Routine.

2.3.3 `taylor_sin`

Diese Funktion implementiert in gewisser Weise die Taylor-Reihe für die Berechnung der Sinus-Funktion. Sie wird in der `init_table`-Routine dazu verwendet um die Interpolationstabelle zu berechnen.

Ausschnitt aus `sinus.asm`

```
taylor_sin:
    call fmod                ; Wert auf ein Intervall
                           ; [0,2*PI[ bringen

fld1
fstp TWORD [faculty]      ; faculty initialisieren

    fld st0                 ; x^2 erzeugen damit das multiplizieren
```

```

    fmul st0, st1          ; in der Schleife an Geschwindigkeit
    fstp TWORD [square_x] ; gewinnt

    fldz
    fstp TWORD [taylor_val] ; taylor_val initialisieren

    fld1
    fstp TWORD [faculty_count] ; faculty_count initialisieren

    fld TWORD [minus_one]
    fstp TWORD [plus_minus_one] ; plus_minus_one mit -1 initialisieren

    ;!! Achtung !! Wenn im Folgenden von Stack die Rede ist,
    ;                ist nicht der Stack der CPU gemeint, sondern
    ;                der Operandenstack der FPU

    while:                ; beim Eintritt in die Iteration steht
                          ; der aktuelle Wert fuer x oben auf
                          ; dem Stack
        ; neues Reihenglied berechnen
    fld st0                ; kopieren von x
    fld TWORD [faculty]    ; aktuellen Wert fuer faculty auf
                          ; den Stack schreiben
    fdivp st1, st0         ; dividiert den aktuellen Wert von
                          ; x durch faculty (x/faculty)

    fld TWORD [minus_one] ; minus_one auf den Stack schreiben
    fld TWORD [plus_minus_one] ; plus_minus_one auf den Stack schreiben
    fmulp st1,st0         ; plus_minus_one wird mit minus_one
                          ; multipliziert
    fld st0                ; neuen Wert fuer plus_minus_one kopieren
    fstp TWORD [plus_minus_one] ; und abspeichern

    fmulp st1, st0        ; je nach Glied wird x/fac mit 1 oder
                          ; minus 1 multipliziert
    fld TWORD [taylor_val] ; speichert taylor_val in st0

    fadd st1, st0         ; bisherige Reihe mit neuem Wert addieren

    fcomp st1             ; bisherige Reihe mit neuer Reihe vergleichen
    fstsw ax              ; das Statuswort in ax abspeichern

```

```

wait                    ; auf die FPU warten
sahf                    ; speichert die hoeherwertigen 8 Bit im
                        ; Statusregister das "Zeroflag" im
                        ; Condition Code der FPU, das beim
                        ; Vergleich gesetzt wird, kann jetzt
                        ; durch das Zeroflag
                        ; in der CPU abgefragt werden
jz end                  ; sollte sich durch das neue
                        ; Reihenglied nichts an der
                        ; urspruenglichen Reihe geaendert
                        ; haben wird die Iteration
                        ; beendet, da die CPU das Ergebnis
                        ; des Reihenglieds intern
                        ; nicht mehr darstellen konnte

fstp TWORD [taylor_val] ; der neue genauere Taylorwert
                        ; wird gespeichert

                        ; die neuen Werte fuer die naechste
                        ; Iteration werden berechnet

; neuen x Wert berechnen
fld TWORD [square_x]    ; square_x auf den Stack schreiben
fmulp st1, st0          ; vorherigen x Wert mit square_x
                        ; multiplizieren

; neuen Facultaetswert berechnen
fld TWORD [faculty]     ; faculty auf den Stack schreiben
fld TWORD [faculty_count] ; faculty_count auf den Stack schreiben
fld1
faddp st1, st0          ; faculty_count inkrementieren

fld st0
fld1
faddp st1, st0          ; faculty_count nochmal inkrementieren

fld st0
fstp TWORD [faculty_count] ; neuen Wert fuer faculty_count speichern

fmulp st1, st0
fmulp st1, st0          ; neuen Fakultaetswert berechnen

```

```

        fstp TWORD [faculty]      ; neuen Fakultaetswert abspeichern
        jmp while                 ; Iteration von neuem beginnen
    end:                            ; Ende der Iteration
        ; x Wert, der noch unten auf dem Stack liegt, entfernen
    fxch st1;                       ; vertauschen von st1 mit st0
    ffree st0                       ; st0 leeren
    fincstp                          ; stack inkrementieren

ret

```

Ausschnitt aus `sinus.asm`

Auf die Funktion `taylor_sin` wird im Folgenden nur oberflächlich eingegangen, da die Funktion ausreichend kommentiert wurde und sich dadurch fast von selbst erklärt.

Der Algorithmus berechnet sukzessive jedes Glied der Taylor-Reihe und addiert sie schrittweise auf. Die Genauigkeit ist dabei nur von der Hardware beschränkt, denn die Funktion rechnet so lange, bis sich zwei aufeinanderfolgende Partialsummenglieder nicht mehr unterscheiden, was bedeutet, dass sie jenseits der vom Computer erfassbaren Genauigkeit sind.

2.3.4 fmod

Wie bereits in der Spezifikation erläutert, ist die Sinus-Funktion eine periodische Funktion mit der Periode $2 \cdot \pi$. Natürlich nutzen wir als Programmierer diese Periode aus. Diese Funktion reduziert das übergebene Argument x auf das Intervall $[0, 2 \cdot \pi[$.

Wir befolgen hier die Formel, die schon in der Spezifikation postuliert wurde.

Ausschnitt aus `sinus.asm`

```

fmod:
    call floor_mode    ; schneidet beim Runden
                       ; den Nachkommateil
    fld st0            ; der Parameter x wird
                       ; kopiert und steht nun
                       ; sowohl in ST0 als auch in ST1

```

```

    fldpi          ; Laedt pi in ST0
    fadd st0, st0  ; in ST0 steht nun der Wert 2*pi
    fdivp st1, st0 ; x/(2*pi)
    frndint       ; rundet ST0 (schneidet
                  ; den Nachkommateil ab)
                  ; --> floor(x/2*pi)
    fldpi          ; Laedt pi in ST0
    fadd st0, st0  ; in ST0 steht nun der
                  ; Wert 2*pi
    fmulp st1,st0  ; --> floor(x/(2*pi))*(2*pi)
    fld st1        ; lade x auf den TOS
    fldz          ; lade 0 auf den TOS
fcompp           ; vergleiche x mit 0
fstsw WORD [statusWordReg]
fwait
mov ax, WORD [statusWordReg]
sahf
ja negative
fsubp st1,st0    ; --> x-floor(x/(2*pi))*(2*pi)
jmp ende
negative: fsubp st1, st0 ; --> x-floor(x/(2*pi))*(2*pi)
fldpi          ; Pi auf den TOS laden
fadd st0, st0    ; 2pi steht auf dem TOS
faddp st1, st0   ; --> x-floor(x/(2*pi))
                  ; *(2*pi)+(2*pi)
ende:          call default_mode ; default-Werte fuer
                  ; die Round-Control
ret

```

Ausschnitt aus `sinus.asm`

Die Funktion ist sehr kurz und kann im Wesentlichen mittels Kommentaren leicht nachvollzogen werden. Zum besseren Verständnis werden wir die Funktion hier noch einmal illustrieren:

```
fct fmod(Real x, Real m)Real:x - floor(x/m) · m
```

Achtung: Diese Funktion ist nur dann richtig, wenn die Zahl $x \in \mathbb{R}$ aus dem Positivitätsbereich \mathbb{R}_+ stammt. Ist sie hingegen aus \mathbb{R}_-

so muss nach der Ausführung der obigen Funktion noch $m \in \mathfrak{R}$ hinzuaddiert werden. Die Formel sieht dann aus wie folgt:

```
fct fmod(Real x, Real m)Real: (x - floor(x/m) · m) + m
```

x stellt in dieser Funktion die Eingabe dar. m wird in unserem Fall $m = 2 \cdot \pi$ sein. Die Gleitkomma-Modulo Operation berechnet also den Rest aus einer Division zweier Zahlen $x \in \mathfrak{R}$ und $m \in \mathfrak{R}$ und das Ergebnis ist im Intervall $[0, m[$.

2.3.5 floor_mode

Intel FPUs bieten drei Modi für Operationen, bei denen gerundet werden muß. In unserer Implementierung werden sowohl die normale Rundung als auch das Abschneiden der Nachkommastellen benötigt. `floor_mode` schaltet die FPU in den "Abschneidemodus".

`floor_mode` kommt im Wesentlichen bei Index-Berechnungen, und in der Funktion `fmod` zum Einsatz.

Ausschnitt aus `sinus.asm`

```
floor_mode:
fstcw WORD [controlWordReg]      ; Kontroll-Wort-Register
                                ; wird gespeichert
wait                               ; die CPU muss auf den
                                ; Coprozessor warten
or WORD [controlWordReg], 0x0C00 ; die Bits 11 und 10
                                ; (Round Control) werden
                                ; beide auf 1 gesetzt was
                                ; bedeutet, dass der
                                ; Nachkommateil vollkommen
                                ; abgeschnitten wird
fldcw WORD [controlWordReg]      ; der veraenderte Wert wird
                                ; ins Kontroll-Wort-Register
                                ; geladen
ret
```

Ausschnitt aus `sinus.asm`

In `floor_mode` wird lediglich das Kontrollwort-Register der FPU gesetzt. Zum Setzen des für uns relevanten “Abschneidemodus” müssen wir die Bits 10 und 11 setzen.

2.3.6 `default_mode`

Diese Routine setzt den Rundungsmodus wieder auf den ursprünglichen Wert zurück. Die FPU rundet also bei der Anwendung eines Rundungsbefehls auf die nächste ganze Zahl.

Ausschnitt aus `sinus.asm`

```
default_mode:
fstcw WORD [controlWordReg]      ; Kontroll-Wort-Register
                                   ; wird gespeichert
wait                               ; die CPU muss auf den
                                   ; Coprozessor warten
and WORD [controlWordReg], 0xF3FF ; die Bits 11 und 10 (Round Control)
                                   ; werden beide auf 0 gesetzt was
                                   ; bedeutet, dass eine Rundung zur
                                   ; naechsten geraden
                                   ; oder ungerade Zahl erfolgt
                                   ; (default Wert)
fldcw WORD [controlWordReg]      ; laedt den Wert in controlWordReg ins
                                   ; Kontroll-Wort-Register
ret
```

Ausschnitt aus `sinus.asm`

Hier gehen wir im Prinzip gleich vor, wie bei der Routine `floor_mode`. Wir setzen hier allerdings die Bits 10 und 11 wieder auf 0.