

Arsinh – 80x86 Assembler

Gruppe 12

Marc Bachmann
Alexander Ringhut
Alexander Scherer

Aufgabenstellung

Gesamtaufgabe:

Das Gesamtprojekt beschaeftigt sich mit der Realisierung von verschiedenen häufig genutzten Gleitkomma-Operationen im 80x86 Assembler.

Teilaufgabe unserer Gruppe:

Die von uns zu bearbeitende Aufgabe besteht darin, die Arcus-Sinus-Hyperbolicus Funktion mit Hilfe von Reihenentwicklung unter alleiniger Zuhilfenahme der Grundrechenarten in einem Assembler-Modul zu implementieren. Dabei soll die Effizienz anhand eines Rahmenprogramm in C durch Zeitmessung der Assembler-Implementierung überprüft werden und mit der Standard-Funktion in C (`asinh()` aus der `<math.h>`) verglichen werden.

Realisierung

Bei der Implementierung des Area Sinus Hyperbolicus gehen wir davon aus, dass die Funktion durch eine Reihe von Summanden darstellbar ist:

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Für die Berechnung sind 4 Fragen zu beantworten:

1. Wie implementiert man das alternierende Vorzeichen?
2. Wie teilt man die Reihe geschickt auf?
3. Wie realisiert man die “kummulative” Multiplikation $(1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1))$ bzw. $(2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n)$?
4. Ist $|x| < 1$?

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

1. Wie implementiert man das alternierende Vorzeichen?

Allgemein: Unser Algorithmus braucht einen Zähler, um zu sehen, wieviele Summanden berechnet werden müssen (n). Die Berechnung findet in einer Schleife statt:

```
for (i=1 ; i<=n ; i++)
{
    ERGEBNIS = ERGEBNIS + SUMMANDEN
    ERGEBNIS = ERGEBNIS - SUMMANDEN
}
```

Antwort:

Man wechselt zwischen Addition und Subtraktion.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

2. Wie teilt man die Reihe geschickt auf?

Zähler 1:

$$1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)$$

Zähler 2:

$$x^{2n+1}$$

Nenner 1:

$$2 \cdot 4 \cdot 6 \cdot \dots \cdot (2n)$$

Nenner 2:

$$(2n+1)$$

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

3. Wie realisiert man die “kumulative” Multiplikation $(1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1))$ bzw. $(2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n)$?

Allgemeines zum Algorithmus:

Die FPU stellt uns 8 Register zur Verfügung, auf die man ähnlich wie auf den Stack zugreifen kann. Dafür stehen eingens Befehle zur Verfügung (sowohl Stack Operatoren als auch mathematische).

Registerbelegung:

Wir belegen vier Register entsprechend der vorgehenden Aufteilung der Reihe: (n = 0 entspricht x, folgende Belegung n=1)

Zähler 1 Register = 1

Zähler 2 Register = x^2

Nenner 1 Register = 2

Nenner 2 Register = 3

Bis auf das Zähler 2 Register (konstant mit dem Wert x^2) werden Zähler 1, und beide Nenner im Laufe der Berechnung um den Wert 2 erhöht.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

3. Wie realisiert man die “kummulative” Multiplikation $(1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1))$ bzw. $(2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n)$?

Zudem speichern wir zusätzlich zum Ergebnis, das Ergebnis des zuvor berechneten Wertes ($n=n-1$) als *Zwischenergebnis* ab.

VOR DER BERECHNUNG:

Damit wir das Zwischenergebnis für die gesamte Berechnung nutzen können müssen wir im Nenner den nicht kumulativen Wert $(2n+1)$ “loswerden”:

Zwischenergebnis $\cdot (2n+1)$

→ Zwischenergebnis weiter verwendbar

WERTE VORBEREITEN:

Bis auf das Zähler 2 Register werden Zähler 1, und beide Nenner im Laufe der Berechnung um den Wert 2 erhöht.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

3. Wie realisiert man die “kumulative” Multiplikation
 $(1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1))$ bzw. $(2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n)$?

BERECHNUNG AN SICH

Zwischenergebnis = Zwischenergebnis
*** Zähler 1 * Zähler 2**
/ Nenner 1 / Nenner 2

4. Ist $|x| < 1$?

```
if ((x > -1) && (x < 1))
```

Falls x außerhalb des definierten Bereichs ist wird NaN zurückgegeben, was durch eine Division durch 0 erzeugt wird.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Realisierungs Idee in C

```

if ((x > -1) && (x < 1))
{
    for (i=1; i<=n; i++)
    {
        Erg=Erg + (2i-1) * x^2
                  / (2i) / (2i+1);

        prep_next_value ();
        [ if (i == n) break; ]
        i++;

        Erg=Erg - (2i-1) * x^2
                  / (2i) / (2i+1);
        prep_next_value ();
    }
}

```

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Lösung

Angenommen x liegt im Definitionsbereich.
Wie werden Fließkommazahlen mit Assembler
berechnet?

Dazu verwendet man die FPU!

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Initialisierung

Wie bereits erwähnt werden die Register passend belegt!

Nenner 2	3	ST0
Nenner 1	2	ST1
Zähler 2	x^2	ST2
Zähler 1	1	ST3
Ergebnis	x	ST4

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Berechnung für n = 0

FLD ST4

	x	
Nenner 2	3	ST0
Nenner 1	2	ST1
Zähler 2	x ²	ST2
Zähler 1	1	ST3
Ergebnis	x	ST4

Mit dem FLD Befehl kann
man sich Werte auf den
TOS der FPU holen

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Call calc_next_value, n=1

CALL calc_next_value		
TEMP	x	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x ²	ST3
Zähler 1	1	ST4
Ergebnis	x	ST5

Der zuvor mit FLD geholte Wert liegt
nun auf dem Stack! Der Rest wandert
nach unten

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

calc_next_value, n=1

FMUL ST0, ST4		
TEMP	x	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x ²	ST3
Zähler 1	1	ST4
Ergebnis	x	ST5

In der calc_next_value
Unterroutine werden die einzelnen
Summanden berechnet.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

calc_next_value, n=1

FMUL ST0, ST3		
TEMP	x*1	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x ²	ST3
Zähler 1	1	ST4
Ergebnis	x	ST5

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

calc_next_value, n=1

FDIV ST0, ST2		
TEMP	↑	ST0
Nenner 2		ST1
Nenner 1		ST2
Zähler 2		ST3
Zähler 1		ST4
Ergebnis		ST5

x*1*x*x
3
2
x ²
1
x

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

calc_next_value, n=1

FDIV ST0, ST1

TEMP	$x * 1 * x * x / 2$	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x^2	ST3
Zähler 1	1	ST4
Ergebnis	x	ST5

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

calc_next_value, n=1

RET

TEMP

$x * 1 * x * x / 2$ **/3**

ST0

Nenner 2

3

ST1

Nenner 1

2

ST2

Zähler 2

x^2

ST3

Zähler 1

1

ST4

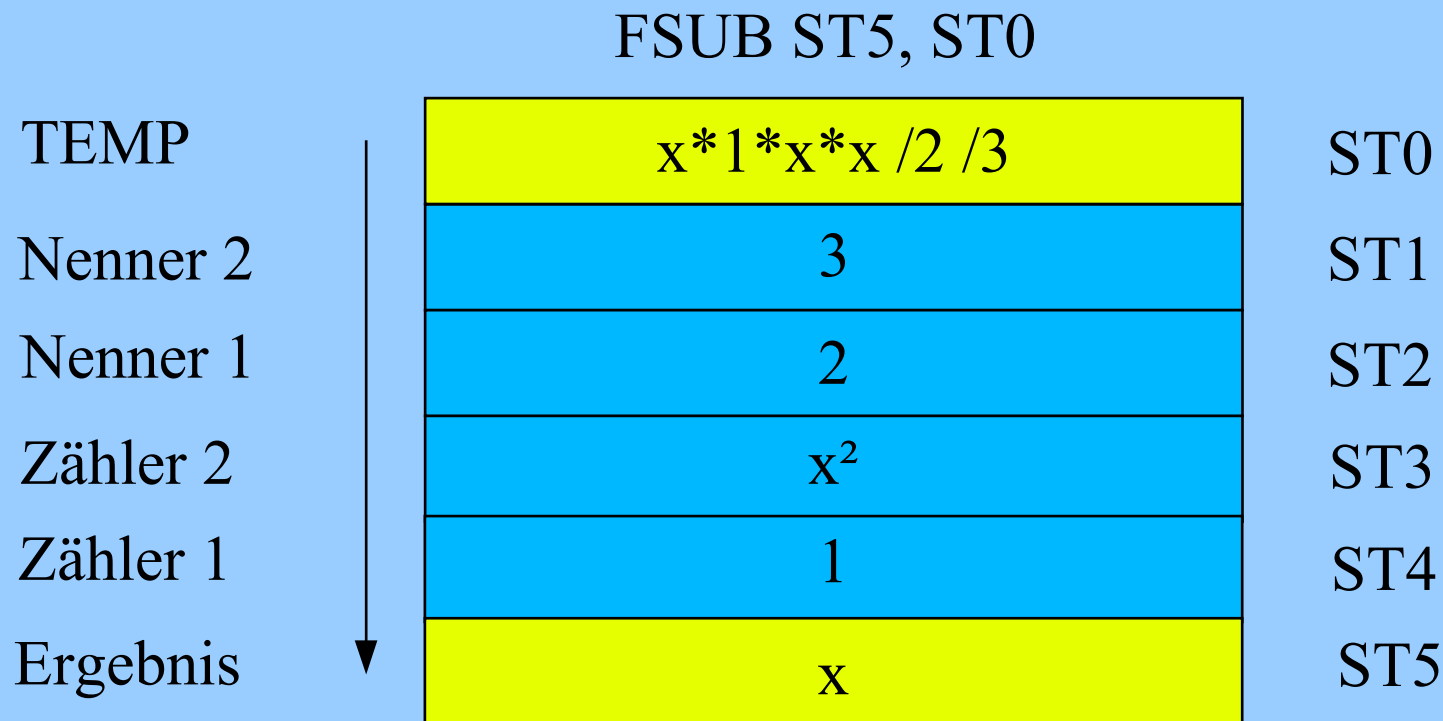
Ergebnis

x

ST5

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Looping, n=1



Der soeben berechnete Wert wird
vom Ergebnis abgezogen
(alternierendes Vorzeichen!)

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Looping, n=1

CALL prep_next_value

TEMP	$x * 1 * x * x / 2 / 3$	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x^2	ST3
Zähler 1	1	ST4
Ergebnis	$x - x * 1 * x * x / 2 / 3$	ST5

Zur weiteren Berechnung müssen die Register neu belegt werden...

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

FMUL ST0, ST1		
TEMP	↑	ST0
Nenner 2		ST1
Nenner 1		ST2
Zähler 2		ST3
Zähler 1		ST4
Ergebnis		ST5

Zur weiteren Berechenbarkeit
müssen wir das $(2n+1)$
“loswerden”!

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

FLD1

	1	
TEMP	$x * 1 * x * x / 2$ 3	ST0
Nenner 2	3	ST1
Nenner 1	2	ST2
Zähler 2	x^2	ST3
Zähler 1	1	ST4
Ergebnis	$x - x * 1 * x * x / 2 / 3$	ST5

Damit die einzelnen Werte erhöht werden können laden wir uns eine 1 auf den FPU-Stack.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

Zweimal: FADD ST5, ST0

ONE	↓	1	ST0
TEMP		$x * 1 * x * x / 2$	ST1
Nenner 2		3	ST2
Nenner 1		2	ST3
Zähler 2		x^2	ST4
Zähler 1		1	ST5
Ergebnis		$x - x * 1 * x * x / 2 / 3$	ST6

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

Zweimal: FADD ST3, ST0

ONE	↓	1	ST0
TEMP		$x * 1 * x * x / 2$	ST1
Nenner 2		3	ST2
Nenner 1		2	ST3
Zähler 2		x^2	ST4
Zähler 1		1+1+1	ST5
Ergebnis		$x - x * 1 * x * x / 2 / 3$	ST6

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

FADD ST2, ST0

ONE	↓	1	ST0
TEMP		$x * 1 * x * x / 2$	ST1
Nenner 2		3	ST2
Nenner 1		2+1+1	ST3
Zähler 2		x^2	ST4
Zähler 1		1+1+1	ST5
Ergebnis		$x - x * 1 * x * x / 2 / 3$	ST6

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

FADDP ST2, ST0

ONE	↓	1	ST0
TEMP		$x * 1 * x * x / 2$	ST1
Nenner 2		3+1	ST2
Nenner 1		$2+1+1$	ST3
Zähler 2		x^2	ST4
Zähler 1		$1+1+1$	ST5
Ergebnis		$x - x * 1 * x * x / 2 / 3$	ST6

Das P im FADDP Befehl bewirkt,
dass die alte Ordnung im Stack
wieder hergestellt wird.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

prep_next_value

RET

TEMP

$x * 1 * x * x / 2$

ST0

Nenner 2

3+1+1

ST1

Nenner 1

2+1+1

ST2

Zähler 2

x^2

ST3

Zähler 1

1+1+1

ST4

Ergebnis

$x - x * 1 * x * x / 2 / 3$

ST5

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Looping, n=2

CALL calc_next_value

TEMP	$x^3 * 1 / 2$	ST0
Nenner 2	5	ST1
Nenner 1	4	ST2
Zähler 2	x^2	ST3
Zähler 1	3	ST4
Ergebnis	$x - x^3 / 2 / 3$	ST5

Anschließend kann der nächste Summand der Reihe berechnet werden (n=2).

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Looping, n=2

FADD ST5, ST0

TEMP		X⁵*1*3 /2 /4 /5	ST0
Nenner 2		5	ST1
Nenner 1		4	ST2
Zähler 2		x ²	ST3
Zähler 1		3	ST4
Ergebnis	↓	x - x ³ /2 /3	ST5

Der soeben berechnete Wert wird
zum Ergebnis aufaddiert
(alternierendes Vorzeichen!)

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Looping, n=2

CALL prep_next_value		
TEMP	$x^5 * 1 * 3 / 2 / 4 / 5$	ST0
Nenner 2	5	ST1
Nenner 1	4	ST2
Zähler 2	x^2	ST3
Zähler 1	3	ST4
Ergebnis	$x - x^3 / 2 / 3 + x^5 * 1 * 3 / 2 / 4 / 5$	ST5

Nun wird der nächste Wert vorbereitet und anschließend berechnet. Die Schleife kann von vorne beginnen.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

WICHTIG

Zwischen zwei Wertberechnungen und am Ende der looping-Schleife muss verglichen werden, alle Werte bis n bereits berechnet wurden:
Der Schleifenzähler (n) wird bei am Anfang des Programms in eines der Register geladen:

MOV ECX, 280

Zwischen zwei Werteberechnungen muss das Register herabgesetzt werden. Falls alle 280 Werte berechnet wurden ($\text{ECX} = 0$) wird das Programm beendet und das Ergebnis in den TOS geladen, damit sich das C-Programm als Rückgabewert holen kann.

DEC ECX
JNZ looping

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Testergebnisse

Insgesamt schlägt die Bibliotheksfunktion die Assemblerfunktion!

Bei ca. 280 Schleifendurchläufen erhalten wir ein auf bis ca. 16 Stellen identisches Ergebnis (bei den meisten Werten).

Nähert sich das übergebene Argument den Intervallgrenzen, so gehen die Ergebnisse auseinander, denn bei gleichbleibender Schleifenanzahl (280) wird Assembler ungenauer. Um identische Ergebnisse zu erhalten muss man die Schleifenanzahl SEHR stark erhöhen, was Zeit kostet!

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

Testergebnisse: Genauigkeit

Summanden	identische Werte (von 39)	identisch im Bereich	identisch bis Nachkommastelle
11	9	+ -0,20	-
21	19	+ -0,45	-
41	27	+ -0,65	-
61	31	+ -0,75	-
81	33	+ -0,80	-
101	35	+ -0,85	6
121	35	+ -0,85	10
141	37	+ -0,90	11
161	37	+ -0,90	11
181	37	+ -0,90	11
201	37	+ -0,90	13
221	37	+ -0,90	13
241	37	+ -0,90	14
261	37	+ -0,90	15
281	alle	+ -0,95	16

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Testergebnisse: Geschwindigkeit

Durchläufe	ASM in ms	C in ms	Asm/C
2	47	36	1,3
5	75	37	2,02
10	107	37	2,89
100	849	37	22,94
200	1671	37	45,16
400	3327	36	92,41

Each sample counts as 0.01 seconds.

%	cumulative	self	
time	seconds	seconds	name
51.16	15.25	15.25	calc_next_value
40.96	27.46	12.21	prep_next_value
6.17	29.30	1.84	looping
0.67	29.50	0.20	asinh_asm
0.60	29.68	0.18	end

Wie man erkennen kann ist C
deutlich schneller als
Assembler.

Die Assembleroutine verbringt
die meiste Zeit in der
calc_next_value Subroutine!

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

asinh aus math.h

```
{
    if(-x == MATH__ sqrt(1+x*x)) return MATH__ log (1/(2 * MATH__
fabs(x)));
    else return MATH__ log(x + MATH__ sqrt(1+x*x) );
}

# define MATH__ ::std::

inline TBCI__ cplx<T>  log (const TBCI__ cplx<T>& z) # define TBCI__
::TBCI::
```

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

asinh aus math.h

```
>>> import math
>>> def asinh(x):
    return math.log(x+(x**2 + 1)**0.5)
>>> asinh(0)
0.0
>>> asinh(1)
0.88137358702
```

$$f(x) = \ln\left(x + \sqrt{x^2 + 1}\right) \qquad g(x) = \operatorname{asinh}(x)$$

$$f(0) = 0 \qquad g(0) = 0$$

$$f(1) = 0.881373587 \qquad g(1) = 0.881373587$$

C berechnet also den asinh nicht über Reihen sondern über ln. Die Werte bleiben die gleichen, doch die C-Funktion gewinnt dadurch an Geschwindigkeit.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} x^{2n+1} \pm \dots \quad \text{für } |x| < 1$$

Entleeren des Stacks

```
end:
  FFREE ST0
  FINCSTP
  FFREE ST0
  FINCSTP
  FFREE ST0
  FINCSTP
  FFREE ST0
  FINCSTP
  FFREE ST0
  FINCSTP
```

VS.

```
FCOMP ST1          ;to clear stack....
FCOMP ST1
FCOMP ST1
FCOMP ST1
FCOMP ST1          ; until just the result is within the stack!
```

Die erste Variante braucht bei der Berechnung von ca 2 Mrd Werten 29,79 Sekunden im Vergleich dazu ist die etwas unelegantere Variante jedoch mit nur 29,65 Sekunden schneller!

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

ASM.ASM

```

GLOBAL asinh_asm          ; declare the function for the c programm to call it

asinh_asm:
;=====
;initialize system
;=====
    PUSH EBP                ; save the basepointer
    MOV EBP,ESP             ; SP := BP

    FINIT                   ; initialize the fpu

    FLD1                    ; load the 1 into the fpu stack
    FCHS                    ; load the -1 into stack

    FLD qword [EBP+8]        ; load the parameter x from C wrapper programm into ST0
                                ; this x is safed during the whole programm
;=====
;check if the c-transmitted x is usable
;=====
    ;check if the |x| < 0

    ;first check -1

    FCOM ST1                ; compare x to -1

    PUSH AX                 ; save AX
    FSTSW AX                ; status word to AX
    SAHF                    ; move ah to flag-register
    POP AX                  ; restore AX
    JBE err_NaN             ; if x <= -1: error NaN (Not a Number)

    ;next check 1
                                ; change -1 to 1
    FXCH ST1                ; move -1 to tos to change sign
    FCHS                    ; load the -1 into stack
    FXCH ST1                ; move x back to tos

    FCOM ST1                ; compare x to 1
    PUSH AX                 ; save AX
    FSTSW AX                ; status word to AX
    SAHF                    ; move ah to flag-register
    POP AX                  ; restore AX
    JAE err_NaN             ; if x >= 1: error NaN (Not a Number)

    ;OK checking is done. TOS now holdes x
    ;start to compute the asinh()

;=====
;initialize the counter with 135 giving us a resonable accuacy
;=====
    PUSH ECX
    MOV ECX, 280            ; init
;=====

```

```

    FXCH ST1                ; move 1 to tos ->
                                ; zaehler1 "register" is initialized with [1]
    FLD1                    ; zaehler2 "register" is initialized with [x]
    FMUL ST0,ST2             ; zaehler2 "register" is initialized with [x\*x{00B2}]
    FLD1                    ; nenner1 "register" is initialized with [1]
    FADD ST0, ST2            ; ST0 <- ST0 + 1 (nenner1 "register" is initialized with [2])
    FLD1                    ; nenner2 "register" is initialized with [1]
    FADD ST0, ST1            ; ST0 <- ST0 + 2 (nenner2 "register" is initialized with [3])
    FLD ST4                  ; tmp_result "register" is initialized with [x]

```

```

looping:
    CALL calc_next_value
    FSUB ST5, ST0            ; Subtracts result - tmp_result
    DEC ECX                  ; decrement the counter
    JZ end
    CALL prep_next_value
    CALL calc_next_value
    FADD ST5, ST0            ; Adds result + tmp_result
    CALL prep_next_value
    DEC ECX                  ; decrement the counter
    JNZ looping              ; loop if we are not allready done

```

```

end:
    POP ECX
    FCOMP ST1                ;to clear stack....
    FCOMP ST1
    FCOMP ST1
    FCOMP ST1
    FCOMP ST1                ; until just the result is within the stack!
    POP EBP                  ; restore BP
    RET                      ; return to C program

```

```

calc_next_value:
    FMUL ST0, ST4            ; tmp_result * zaehler1
    FMUL ST0, ST3            ; tmp_result * zaehler2
    FDIV ST0, ST2            ; tmp_result / nenner1
    FDIV ST0, ST1            ; tmp_result / nenner2
    RET

```

```

prep_next_value:
    FMUL ST0, ST1            ; tmp_result * nenner2
    FLD1                    ; Adds 1 to TOS
    FADD ST5, ST0            ; zaehler1 + 1
    FADD ST5, ST0            ; zaehler1 + 1 => zaehler1 + 2
    FADD ST3, ST0            ; nenner1 + 1
    FADD ST3, ST0            ; nenner1 + 1 => nenner1 + 2
    FADD ST2, ST0            ; nenner2 + 1
    FADDP ST2, ST0 ;nenner2 + 1 => nenner2 +2, FADDP for the 1 of TOS to be popped, and the
                                original order to be restored!
    RET

```

```

;=====
;helper function to "generate" NaN
;=====
err_NaN:

```

```

    FINIT
    FLDZ                    ; ST0 <- 0
    FDIV ST0, ST0           ; ST0 <- 0/0 (0/0 is not defined and therefore returns NaN)
    POP EBP                 ; restore BP
    RET                    ; return to C program

```

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

MAKEFILE

```
asinh: asinh.o asm.o
```

```
gcc -Wall -g -lm asinh.o asm.o -o asinh
```

```
asinh.o: asinh.c asm.o
```

```
gcc -c -Wall asinh.c
```

```
asm.o: asm.asm
```

```
nasm -f elf asm.asm
```

Die Option -f elf gibt an, daß der Assembler eine Objektdatetei asm.o im ELF (Extended Linker Format) erzeugen soll.

$$\operatorname{arsinh}(x) = x - \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \dots + (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) x^{2n+1}}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2n(2n+1)} \pm \dots \quad \text{für } |x| < 1$$

C-RAHMEN

```
# include <sys/wait.h>
# include <signal.h>
# include <sys/time.h>

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
#include <math.h>      /* for asinh() */

extern double asinh_asm ( double );

struct timer
{
    int milliseconds;
    int seconds;
    int minutes;
    int hours;
    int days;
};

struct timer timer1;
struct timer timer2;
struct timer timer3;
struct timer timer0;

struct timer set_timer (struct timer timerx)
{
    timerx.milliseconds++;
    if (timerx.milliseconds == 100)
    {
        timerx.milliseconds = 0;
        timerx.seconds++;
    }
    if (timerx.seconds == 60)
    {
        timerx.seconds = 0;
        timerx.minutes++;
    }
    if (timerx.minutes == 60)
    {
        timerx.minutes = 0;
        timerx.hours++;
    }
    if (timerx.hours == 24)
    {
        timerx.hours = 0;
        timerx.days++;
    }
    return timerx;
}

void signalhandle1 ()
{
    timer1 = set_timer (timer1);
}
void signalhandle2 ()
{
    timer2 = set_timer (timer2);
}
void signalhandle3 ()
{
    timer3 = set_timer (timer3);
}
```

```
int start_prof ()
{
    timer1=timer2=timer3=timer0;
    struct itimerval val;
    signal (SIGALRM,signalhandle1);    /*
handler for SIGALRM */
    signal (SIGVTALRM,signalhandle2);  /*
handler for SIGALRM */
    signal (SIGPROF,signalhandle3);    /*
handler for SIGALRM */

    val.it_value.tv_sec = 0;

    val.it_value.tv_usec = 1;
    val.it_interval.tv_sec = 0;
    val.it_interval.tv_usec = 1;

    if ((setitimer(ITIMER_REAL,&val,0) == 0)
    && (setitimer(ITIMER_VIRTUAL,&val,0) ==
0) && (setitimer(ITIMER_PROF,&val,0) == 0))
        return (0);
    else return (-1);
}

int stop_prof ()
{
    struct itimerval val;
    if ((getitimer (ITIMER_REAL,&val) == 0) &&
        (getitimer(ITIMER_VIRTUAL,&val) == 0)
    && (getitimer(ITIMER_PROF,&val) == 0))
        return (0);
    else return (-1);
}

void print_prof ()
{
    /* printf ("REALE ZEIT: \n"); */
    if (timer1.days !=0)
        printf ("%i Tage, ", timer1.days);
    if (timer1.hours !=0)
        printf ("%i Stunden, ", timer1.hours);
    printf ("%i Minuten, ", timer1.minutes);
    printf ("%i Sekunden, ", timer1.seconds);
    printf ("%i Millisekunden.\n", timer1.
milliseconds);
}
```

```
int main ()
{
    double erg_c;
    double erg_asm;
    double eingabe;
    printf ("\n*****");
    printf ("\n* Please enter x-value *\n" );
    printf ("*****\n");
    printf ("\nx = " );
    scanf ("%le", &eingabe);
    start_prof ();
    erg_asm=asinh_asm (eingabe);
    stop_prof ();
    printf ("To calculate asinh(%lf) = %.15lf the ASM-
Routine took:\n", eingabe, erg_asm);
    print_prof ();

    start_prof ();
    erg_c=asinh (eingabe);
    stop_prof ();
    printf ("To calculate asinh(%lf) = %.15lf the C-Routine
took:\n", eingabe, erg_c);
    print_prof ();

    if ((erg_c - erg_asm) == 0.0)
        printf ("there's no difference between c and nasm
calculation! \n\n");
    else
        printf ("\n the values differ (result in c - result
in nasm): %.15lf \n\n", (erg_c-erg_asm));

    return (0);
}
```