Praktikum – Rechnerarchitektur

Assembler – Projekt

Spezifikation

G53: Adrian Kögl

Daniel Osipishin

Marc Sinner

Inhaltsverzeichnis

[Aufgabenverteilung 3](#_Toc513553708)

[Abgabetermine: 3](#_Toc513553709)

[Einzelaufgaben 3](#_Toc513553710)

[Aufgabenbeschreibung 4](#_Toc513553711)

[Ist – Analyse 4](#_Toc513553712)

[Soll – Analyse 4](#_Toc513553713)

[Erfolgskriterium 4](#_Toc513553714)

[Lösungsansätze 5](#_Toc513553715)

[Annäherung mittels Newton-Verfahren 5](#_Toc513553716)

[Verworfener Lösungsansatz: Lookup-Table + Interpolation 5](#_Toc513553717)

# Aufgabenverteilung

Adrian Kögl Projektleiter

Daniel Osipishin Verantwortlicher Dokumentation

Marc Sinner Verantwortlicher Vortrag

# Abgabetermine:

Spezifikation 13.05.2018

Implementierung 17.06.2018

Ausarbeitung 08.07.2018

Vortrag 23.07 - 03.08.2018

Generell: Materialien spätestens vor dem letzten Mittwoch vor Abgabe fertig haben (falls es nötig ist, den Tutor zu fragen)

# Einzelaufgaben

Daniel Osipishin Erstellung einer TODO-Liste der zu implementierenden Funktionen mit genau spezifizierten Inputs und Outputs (13.05.18)

Marc Sinner Erstellung und Verwaltung eines Gits

Alle zusammen Implementierung einzelner Funktionen sowie des Gesamtprogramms (12.06.18) – genauere Aufgabenverteilung nach fertiger TODO-Liste

# Aufgabenbeschreibung

Realisierung der Funktion in Assembler. Sowohl der Input x, als auch der Output y sind vom Datentyp float (32 Bit). Dabei ist die Nutzung der FPU nur auf Addition, Subtraktion, Multiplikation, Division, Negation und Datenverwaltung eingeschränkt.

Außerdem, Implementierung eines C-Rahmenprogramms zum Testen der Assembler-Funktion und deren Laufzeit.

## Ist – Analyse

Ein Lösungsansatz ist bereits vorhanden (siehe „Lösungsansätze“).

Der Lösungsansatz ist durch ein Java-Programm als funktionsfähig bewiesen.

Implementierung wurde noch nicht angefangen.

## Soll – Analyse

Zur Implementierung muss die Gesamtheit der Funktionen aus dem Lösungsansatz realisiert werden. Dabei soll die Nutzung von Realisierungs- und Laufzeitsaufwendiger Funktionen wie oder vermieden werden.

## Erfolgskriterium

Der Projekterfolg ist objektiv messbar anhand von:

* Laufzeit der Funktion
* Ergebnisgenauigkeit
* Speicherverbrauch

Idealerweise soll die Funktion die Genauigkeit der float-Mantisse komplett ausnutzen.

# Lösungsansätze

## Annäherung mittels Newton-Verfahren

Also, um y auszurechnen, muss man die Nullstelle nach y der Gleichung finden. Diese kann mittels Newton-Verfahren ermittelt werden.

1. Wähle ein , was möglichst nah an der Nullstelle liegt (funktioniert aber mit beliebigem ). Ein passendes kann einer vorzeitig angefertigten Lookup-Table entnommen werden.
2. Benutze die folgende Formel, bis die erwünschte Genauigkeit erreicht ist

Dabei sind und . x ist der Input und dadurch zur Laufzeit konstant. cosh und sinh lassen sich dabei mittels Taylor-Reihen ausrechnen (bis zur erwünschten Genauigkeit):

Man sieht, dass an keiner Stelle in diesem Algorithmus Rechenoperationen verwendet werden, die nicht gegeben sind (+, -, \*, /), wobei Potenzieren sehr einfach zu implementieren ist. Genaue „Tiefen“ vom Newton-Verfahren, sowie von Taylor-Reihen, die die erwünschte Genauigkeit liefern, sind noch zu bestimmen.

## Verworfener Lösungsansatz: Lookup-Table + Interpolation

Idee: Funktionswerte an bestimmten Stellen speichern und, falls verlangt, an Stellen zwischen der gespeicherten Werten durch Interpolation annähern.

Interpolationsmethoden sind entweder nicht genau genug (lineare Interpolation), oder Laufzeitsaufwendig (polynomielle Interpolation), oder schwer zu implementieren (spline/logarithmische Interpolation). Newton-Verfahren ist genauer, einfacher, (meistens) schneller, verlässt sich nicht auf Lookup-Tables. Meistens wird Interpolation für unbekannte Funktionen mit bekannten Werten benutzt. In diesem Fall ist die Funktion genau bekannt, was genaue Berechnungsmethoden erlaubt, z.B. Newton-Verfahren.