02.07.2016

Praktikum: Einführung in die Rechnerarchitektur

IN0005

Sommersemester 2016

ENTWICKLER DOKUMENTATION

zu

PROJEKT 1

-

Berechnung des ln(x)

Gruppe 15

Benedikt Thoma

Hendrik Pauthner

Michael Schöffmann

Beschreibung

Das mit Atom in ASM und C geschriebene Programm berechnet für eine spezifizierbare Anzahl an Eingaben den natürlichen Logarithmus. Dies wird für Vergleichbarkeit an Genauigkeit und Laufzeit einerseits in C und andererseits in der selbst erstellten Assemblerfunktion getan. Die Ergebnisse werden zusammen mit den Laufzeiten in einer Tabelle für jede Eingabe einzeln ausgegeben.

Input

Das kompilierte Programm erhält über die Kommandozeile mindestens eine Zahl als Input, welche die Anzahl der Inputs spezifiziert. Ist die Zahl 0, so wird ein Beispielprogramm ausgegeben. Negative Zahlen sind nicht zulässig. Gibt man dem Programm als Input eine Zahl zum Berechnen, die nicht innerhalb des Definitionsbereichs des ln(x) liegt, ist das Ergebnis NaN.

Umsetzung

Parsing

Das C Rahmenprogramm parsed die Argumente des Programms. Ist die erste Zahl 0, so erzeugt es eine Liste mit Beispielwerten, für die der ln berechnet wird, sonst parsed es diese Werte aus den Arguments. Ist die Zahl nicht 0, so wird eine Liste erzeugt, in die die Inputs als floats abgespeichert werden. Der Rest ist für beide Fälle gleich.

Berechnen der C-Ergebnisse

Die Liste der Inputs wird in einer for-Schleife durchlaufen. Dabei wird bei jedem Durchlauf die Zeit vor der Berechnung und nach der Berechnung gemessen. Die Laufzeiten werden durch (float)(ende-anfang) berechnet und ebenfalls in eine eigene Liste gespeichert. Das Ergebnis der C-Funktion wird in eine eigene Liste gespeichert.

Berechnen der ASM-Ergebnisse

Die Liste der Inputs wird in einer for-Schleife durchlaufen. Dabei wird bei jedem Durchlauf die Zeit vor der Berechnung und nach der Berechnung gemessen. Die Laufzeiten werden durch (float)(ende-anfang) berechnet und in eine eigene Liste gespeichert. Für das Berechnen der ASM-Ergebnisse wird für jeden Input die Funktion calcasm(float input) aufgerufen. Diese setzt die globale Variable addressInput auf input und ruft die Assemblerroutine auf.

Die Assemblerroutine holt sich den Input über die globale Variable und speichert ihn in den FPU-Stack. Die Register werden mit den der Tabelle entsprechenden Werten initialisiert.

EAX	0, später die Statuswörter der FPU
EBX	Adresse der ln-Tabelle
ECX	Adresse der Bruch-Tablle
EDX	Grenze m für Prec-Schleifendurchläufe
ESI	Laufvariable l
EDI	Flag für Input < 1

Vergleiche in der FPU funktionieren, in dem man ST0 mit der passenden Zahl vergleicht, sich dann das Statuswort in ax lädt, das richtige Bit aus ax verundet und ax dann mit 0 oder 1 vergleicht. Alle Vergleiche im Folgenden funktionieren so, außer es ist die Rede von einem „normalen Vergleich".

Das Programm vergleicht ST0 mit 0, um alle Eingaben <= 0 zu filtern. Ist die Eingabe <= 0, so springt das Programm zu enderror und speichert in die globale Variable addressResult NaN.

Ist der Input > 0, vergleicht das Programm ST0 mit 1, um Eingaben < 1 und = 1 zu filtern. Ist die Eingabe = 1, springt das Programm zu endinpone und speichert in addressResult 0. Ist die Eingabe < 1, dann wird das Flag in EDI auf 1 gesetzt und man springt zu initnegativ. In initnegativ wird das Inverse von ST0 berechnet und in die eigentliche Berechnung gesprungen. Das funktioniert, da $-ln\left(\frac{1}{x}\right) = ln(x)$ für 0 < x < 1.

Für die nun in die richtige Form gebrachten Inputs wird dieser Algorithmus verwendet, um den ln(x) zu berechnen:

1. if x == e: jump endone, else
2. if x < e: jump 4
3. while x > e: x = x / e, inkrementiere z
4. while l < m: if x == 1: jump endnormchecknegativ, sonst teile x durch den größten Eintrag der ln-Tabelle < x und addiere den passenden Eintrag der Bruch-Tablle auf z, inkrementiere l
5. jump endnormchecknegativ

In endone wird 1 in addressResult gespeichert und das Programm returned.

In endnormchecknegativ wird ein normaler Vergleich von EDI und 1 durchgeführt.
Wenn EDI = 1, dann wird ST0 mit -1 multipliziert und das Flag in EDI auf 0 gesetzt. Anschließend wird in endnorm gesprungen. Ist EDI = 0 wird sofort in endnorm gesprungen.

In endnorm wird das in der FPU errechnete Ergebnis an die Stelle von addressResult gespeichert und in das C-Programm zurückgesprungen.

Die in der Assemblerfunktion errechneten Ergebnisse werden im C-Programm in eine eigene Liste gespeichert.

Ausgabe der Ergebnisse

Die Listen werden in einer for-Schleife durchlaufen und die Einträge in der Reihenfolge Input, C-Result, ASM-Result, C-Time, ASM-Time für jeden Durchlauf in einer einzelnen Tabelle ausgegeben.

Modifizierungsmöglichkeiten

Genauigkeit: Die Genauigkeit der Berechnung des ln(x) lässt sich beliebig steigern, indem mehr Werte in kleineren Schritten in die Lookuptabellen eingetragen werden. Allerdings geht dies auf Kosten der Laufzeit, da für jeden Wert ein Schleifendurchlauf hinzukommt.

Laufzeit: Die Laufzeit lässt sich an einigen Stellen optimieren. Einige der Vergleiche könnten außerhalb der FPU stattfinden, was der Laufzeit zugutekommt. Einige Berechnungen wie zum Beispiel das Multiplizieren mit -1 lassen sich optimieren. Des Weiteren kann man sich die C-Routine in Assembler anschauen und einige der Optimierungen des C-Compilers adaptieren. Das Programm könnte man auch auf 64-bit implementieren.

Wichtige Begriffe

Lookup-Tabelle	Eine Tabelle mit Einträgen, von vorberechneten Werten. Kann zur Berechnung von komplizierteren mathematischen Funktionen benutzt werden um Berechnungszeit zu sparen.
NaN	Not a Number. Ein Zahlenwert der zur Kennzeichnung von nicht zulässigen mathematischen Operationen genutzt wird.