# 64-041 Übung Rechnerstrukturen WS 15/16

## Aufgabenblatt 4 Ausgabe: 04.11., Abgabe: 11.11. 24:00

| Gruppe | |
|---|---|
| Name(n) | Matrikelnummer(n) |
| | |

### Aufgabe 4.1 (Punkte 5+5+5)

*Gleitkommazahlen*: Normalisieren Sie die folgenden Gleitkommazahlen unter Beibehaltung der jeweiligen Basis, wobei die erste von Null verschiedene Stelle der Mantisse unmittelbar links vom Komma stehen soll. In den folgenden Aufgaben ist in den Klammern jeweils erst die Mantisse und dann der Exponent angegeben, und die Basis als Suffix hinter den Klammern:

(a) $(341,14 \mid 4)_{10}$

(b) $(-1\,1001,11 \mid 1101\,0010)_2$

(c) $(-0,003D1 \mid 1A)_{16}$

### Aufgabe 4.2 (Punkte 5+5+5+5)

*Gleitkommazahlen*: Wandeln Sie folgende Dezimalzahlen in Gleitkommazahlen einfacher Genauigkeit gemäß IEEE 754 um. Es genügt dabei, wenn Sie die acht höchstwertigsten Bit der Mantisse angeben:

(a) 6

(b) $-8$

(c) $1,75$

(d) $25,125$

### Aufgabe 4.3 (Punkte 10 + 10 + 10 + 10)

*Arithmetische Operationen mit Gleitkommazahlen*: Gegeben seien sie beiden folgenden einfachgenauen Gleitkommazahlen gemäß IEEE 754, wobei von der Mantisse nur die oberen vier Bit angegeben sind, alle anderen Bits sind 0:
$A = (1 \mid 1000\,0001 \mid 1110)_2$ und $B = (0 \mid 1000\,0000 \mid 1100)_2$. Das Zeichen $\mid$ dient hier nur zur besseren Lesbarkeit und Trennung der einzelnen Felder. Berechnen Sie ohne Umwandlung

ins Dezimalsystem $A + B$, $A - B$, $A \cdot B$ und $(A - B)/(A + B)$. Die Ergebnisse sollte wieder gemäß IEEE 754 dargestellt sein. Geben Sie jeweils die einzelnen Rechenschritte an.

## Aufgabe 4.4 (Punkte 5+5+15)

*Vergleich von Gleitkommazahlen*: In Java und C funktionieren die arithmetischen Vergleichsoperationen wie <, <=, >, >= sowie == und != wie erwartet.

(a) Warum ist der direkte Vergleich zweier (auf unterschiedlichen Wegen berechneten) Gleitkommazahlen mittels

```
double a = ...
double b = ...
if (a == b) { ... } else { ... }
```

trotzdem keine gute Idee?

(b) Etwas besser ist der folgende Ansatz:

```
final double EPSILON = ... // z.B. 0.000001
double a = ...
double b = ...

if (Math.abs( a - b ) < EPSILON)
{ ... } // almost equal
else
{ ... } // not equal
```

Begründen Sie, wie Sie abhängig von den Wertebereichen der Variablen $a$ und $b$ sowie der erreichbaren Genauigkeit (hier Datentyp `double`) einen geeigneten Wert von $\epsilon$ wählen.

Dabei sind die von der Anwendung her erreichbaren Genauigkeiten zu berücksichtigen. Zum Beispiel sind mit einem Zollstock gemessene Längen vielleicht nur auf einen halben Millimeter genau, mit einer Mikrometerschraube gemessene Dicken eines Blechs auf einen Mikrometer, und Entfernungsangaben auf Straßenschildern vielleicht auf $\pm 1$ km genau.

(c) Funktioniert der unter (b) entwickelte Ansatz auch, wenn die Variablen $a$ und $b$ im Verlauf der Rechnung Werte sehr unterschiedlicher Größenordnungen (z.B. zunächst $a \approx b \approx 10^{15}$, später aber $a \approx b \approx 10^{-7}$) annehmen können?

Geben Sie ein Verfahren an, dass bei fest gewähltem $\epsilon$ auch für Variablen mit fast gleichen Werten in sehr unterschiedlichen Wertebereichen funktioniert. Benutzen Sie ggf. die Funktionen `Math.abs(a)` für den Absolutwert der Variable $a$, `Math.max(a,b)` für die größere der beiden Werte $a$ und $b$, und `Math.isNaN(a)` für den Test auf den Wert `NaN`:

```
 /* returns 1 if two double-precision numbers are almost equal,
    considering their scale, and 0 otherwise.  */
int AlmostEqual( double a, double b ) {
  ... // your code here
}
```