64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 5 Ausgabe: 16.11., Abgabe: 23.11. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 5.1 (Punkte 10+10+10+10)

Arithmetische Operationen mit Gleitkommazahlen: Gegeben seien die beiden folgenden einfach genauen Gleitkommazahlen gemäß IEEE 754. Von der Mantisse sind jeweils nur die oberen acht Bit angegeben, alle anderen Bits sind 0. Zur besseren Lesbarkeit wird das Zeichen „|“ als Feldtrenner in dem Bitstring benutzt (*s eeee eeee mmmm mmmm*):

$A = (1 \mid 1000\,0010 \mid 0011\,0000)_2$ und
$B = (0 \mid 1000\,0001 \mid 0110\,0000)_2$

Berechnen Sie ohne Umwandlung in das Dezimalsystem die folgenden Ausdrücke. Alle Ergebnisse sollen wieder als IEEE 754 Zahlen (wie oben) dargestellt werden. Geben Sie dabei immer auch die einzelnen Rechenschritte an.

(a) $A + B$

(b) $A - B$

(c) $A \cdot B$

(d) $(A - B) \,/\, (A + B)$

Aufgabe 5.2 (Punkte 5+5+10)

Gleitkomma-Rundung: Wir betrachten ein Gleitkommaformat im Dezimalsystem mit zwei Stellen für den Exponenten und vier Nachkommastellen für die Mantisse. Addieren Sie die beiden Gleitkommazahlen und geben Sie sowohl die Zwischenrechnungen, als auch das normalisierte und gerundete Ergebnis an (dezimal).

$8,617 \cdot 10^5 + 9,9443 \cdot 10^7$

Führen Sie diese Berechnung zweimal, mit unterschiedlichen Rundungsstrategien, durch:

(a) Mit einmaliger Rundung am Ende nach der Normalisierung.

(b) Bei der Berechnung werden alle Zahlen, auch die Zwischenergebnisse, auf vier Nachkommastellen gerundet.

(c) Welches Verfahren ist vorzuziehen? Beziehungsweise, was wäre das sinnvollere Rundungsverfahren? Begründen Sie Ihre Antwort.

Aufgabe 5.3 (Punkte 10+10+5)

ULP: Um zu untersuchen, wie sich Rundungsfehler bei der Gleitkomma-Arithmetik auswirken, betrachten wir folgendes Programm, das in einer Schleife immer wieder den Wert `0,1`, der sich in binärer Gleitkommadarstellung ja nicht exakt darstellen lässt, aufaddiert.

```
1  // Test floating point arithmetic...
2  public class aufg05_3 {
3    public static void main( String args[] ) {
4      // calculate the sum of 1E9 unprecise numbers
5      int n = 0;
6      float sum;
7      float limit = 1.0E8f;
8      for( sum = 0; sum < limit; sum += 0.1 ) {
9        n++;
10 //     if ((n % 100000) == 0) {
11 //       System.out.println("n="+ n + " sum="+ sum + " target="+ (n*0.1));
12 //     }
13     }
14     System.out.println("sum is " + sum + " compared to " + (n*0.1));
15   }
16 }
```

(a) Beschreiben Sie kurz, was das Programm tut. Was erwarten Sie (ungefähr) als Ausgabewert des Programms? Wie viele Iterationen sollten am Ende in der `for`-Schleife durchlaufen worden sein?

(b) Was passiert tatsächlich? Warum?
Tipp: Es kann helfen, die auskommentierten Codezeilen 10 . . . 12 wieder zu aktivieren.

(c) Schreiben Sie das Programm so um, dass es den ursprünglich angedachten Zweck erfüllt.

Aufgabe 5.4 (Punkte 10+5)

Zeichensatzcodierung: Entschlüsseln Sie mit Hilfe der Vorlesungsunterlagen den folgenden hexadezimal codierten Text nach ISO-8859-1.

```
44 69 65 20 41 75 66 67 61 62 65 20 77 61 72 3A 0A 09 28 61 29 20 57 69
65 20 73 69 65 68 74 20 64 65 72 20 54 65 78 74 20 61 75 73 3F 0A 75 6E
64 20 6A 65 74 7A 74 20 6C 69 65 67 74 20 64 69 65 20 4C F6 73 75 6E 67
20 76 6F 72 20 49 68 6E 65 6E 2E 2E 2E 0A
```

(a) Wie sieht der Text aus? Notieren Sie dazu die Textdarstellung (mit Steuerzeichen).

(b) Was verrät Ihnen der Text über den Rechner mit dem er erstellt worden ist?