Prof. Dr. Burkhard Stiller, Universität Zürich, Binzmühlestrasse 14, CH-8050 Zürich
Telefon: +41 44 635 6710, Fax: +41 44 635 6809, stiller@ifi.uzh.ch
Martin Waldburger, Telefon: +41 44 635 4304, waldburger@ifi.uzh.ch
Fabio Hecht, Telefon: +41 44 635 7129, hecht@ifi.uzh.ch

Übungen zu Informatik 1

Technische Grundlagen der Informatik - Übung 10

Ausgabedatum: 21. November 2011
Besprechung: Übungsstunden in der Woche ab dem 28. November 2011

1) Implementation einer Gleitkommazahl

Schreiben Sie ein Javaprogramm, welches eine IEEE 754 Gleitkommazahl mit veränderbarer Anzahl von Bits für die Darstellung von Mantisse und Charakteristik implementiert.
Für Informationen über diesen Standart finden Sie eine Einführung auf
http://de.wikipedia.org/wiki/IEEE_754

Das Programm sollte aus zwei Klassen bestehen, einem TestDriver und einer CustomFloat Klasse. Der TestDriver wird gebraucht um einen CustomFloat zu erzeugen und die Methoden auf dem Objekt aufzurufen (siehe letzte Seite).
Die CustomFloat-Klasse erstellt in internen Variabeln eine Gleitkommazahl basierend auf den beiden Konstanten für Mantisse und Exponent. Beim Erstellen erhält CustomFloat einen double womit dann Vorzeichen, Mantisse und Charakteristik (basierend auf den Konstanten für die Mantisse und Charakteristik) berechnet werden. Die Methode getBits() der Klasse CustomFloat liefert einen long mit der ganzen Gleitkommazahl als binäre Umwandlung. Eine zweite Methode der Klasse ist die printBits(), welche bereits implementiert ist. Die Form der Ausgabe hat dieses Aussehen:

VZ:1 Charakteristik:1111111 Mantisse:10101000111101011100000

Eine weitere bereits implementierte Methode ist die toDouble()-Methode. Diese nimmt die berechneten Werte für das Vorzeichen, Charakteristik und Mantisse und berechnet die daraus entstehende Gleitkommazahl.
Erklären Sie in wenigen Worten, weshalb die Ungenauigkeiten zwischen der Ausgangszahl und der errechneten Gleitkommazahl zustandekommen.

Hinweise:

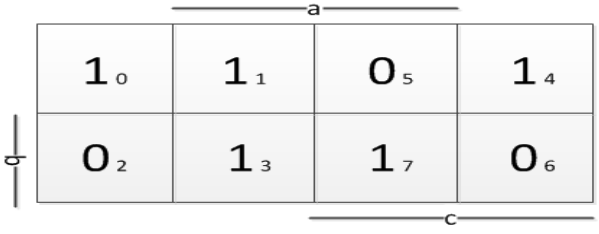
Mathematische Funktionen mit der Klasse Math.
Bsp: Potenzen: Math.pow(2,4)
Betrag: Math.abs(-23);

Bitweise Operationen:
Man kann Bitfolgen mit UND (&) und ODER (|) verknüpfen.
Bitshifts werden mit "<<" oder ">>" erreicht.
Bsp: 2 << 2 ergibt 8

Hinweise zum Framework auf den letzten Seiten!

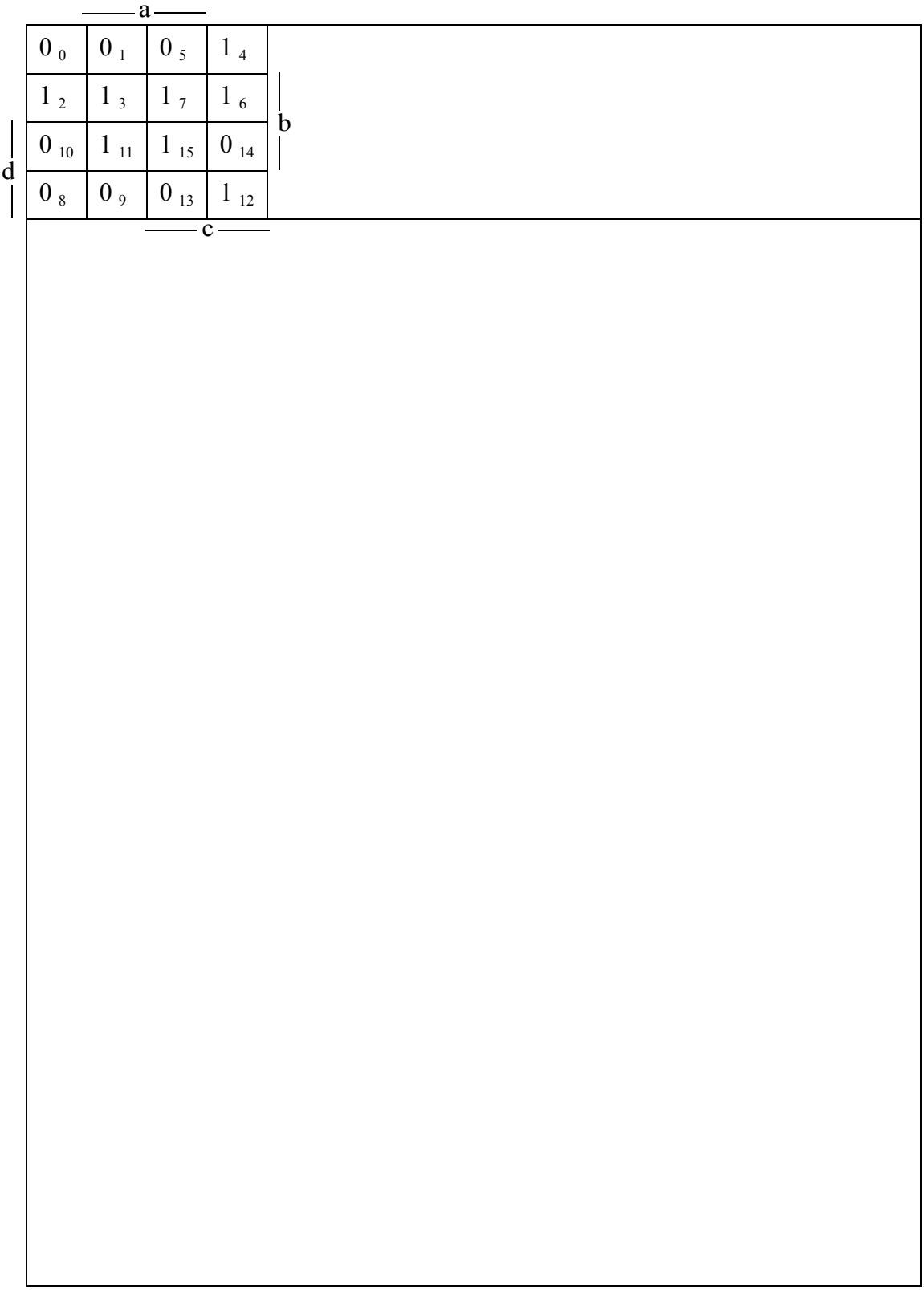
2) KV-Diagramme

2.1) Geben ist folgendes KV-Diagramm. Kreuzen Sie jeweils die richtige Antwort zu folgenden Aussagen an und begründen Sie ihre Antwort.



	Aussage	Richtig	Falsch
a)	Der Minterm bei Index 1 entspricht dem Ausdruck $a\bar{b}c$.		
b)	Benachbarte Felder eines KV-Diagramms unterscheiden sich nur in einer Eingangsvariablen.		
c)	Das KV-Diagramm enthält vier Primimplikanten.		
d)	Das KV-Diagramm beschreibt nur eine disjunktive Minimalform die dem Ausdruck $a\bar{c} \vee \overline{ab} \vee ab$ entspricht.		

2.2) Lesen Sie alle Minterme und Maxterme aus dem untenstehendem KV-Diagramm heraus. Bestimmen Sie anschliessend alle Primimplikanten/Primimplikate sowie die disjunktive/ konjunktive Minimalform bestehend aus möglichst wenigen Primimplikanten/Primimplikate.[1]



1. Auf der Webseite http://ti.itec.uka.de/KVD/index.html finden Sie ein Applet, mit welchem Sie den Umgang mit KV-Diagrammen und die Berechnung der versch. Minimalformen üben können.

Hinweise zum Javaprogramm:

```
class CustomFloat {

        static final int NUM_BITS_MANTISSE=11;
        static final int NUM_BITS_CHARAKTERISTIK=4;

    long mantisse;           //supports max 64 bit with long
    int charakteristik;       //supports max 32 bit with int
    int vz;

    public CustomFloat(double n) {

        System.out.println("n="+n);

        //TODO: calculation vz (sign)

        if (n==0) {
                this.mantisse=0;
                this.charakteristik=0;
        }
        else {

             //TODO: calculation of mantisse and charakteristik


         }
    }

    long getBits() {
        //TODO: returns vz, charakteristik, mantisse in a long
          //(bits to the left set to zero)

    }

    void printBits() {
        //outputs VZ:X Charakteristik:YYY Mantisse:ZZZZZ
        System.out.println("VZ:" + this.vz+
        " Charakteristik:"+Integer.toBinaryString(this.charakteristik) +" Mantisse:"+
        Long.toBinaryString(this.mantisse));
    }

    double toDouble() {
                double output=1;

                for (int i=0; i<NUM_BITS_MANTISSE; i++) {
                        if (0<(this.mantisse & (long)(Math.pow(2,NUM_BITS_MANTISSE-i)))) {
                                output += Math.pow(2, -(i));
                        }
                }

                int bias = (int) Math.pow(2, NUM_BITS_CHARAKTERISTIK-1)-1;

                output *= Math.pow(-1, this.vz) * Math.pow(2,this.charakteristik-bias);
                return output;
    }

}
```

```java
public class TestDriver {

    public static void main(String[] args) {


        //Results produced with the following constants
        //static final int NUM_BITS_MANTISSE=11;
        //static final int NUM_BITS_CHARAKTERISTIK=4;

        CustomFloat cf = new CustomFloat(-15.5);
        cf.printBits();
        System.out.println(cf.toDouble());  //prints -15.5

        cf = new CustomFloat(1.0);
        cf.printBits();
        System.out.println(cf.toDouble());  //prints 1.0

        cf = new CustomFloat(1.666);
        cf.printBits();
        System.out.println(cf.toDouble());  //prints 1.6650390625 (imprecise!)

        cf = new CustomFloat(66.9);
        cf.printBits();
        System.out.println(cf.toDouble());  //prints 66.875 (imprecise!)


    }

}
```