Lösungen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
SS 03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Lösung Übungszettel 4

1 Aufgabe 1: Addition und Subtraktion im Zweierkomplement

```
#include <stdio.h>

//Ein Bitfeld fuer die Darstellung einer 16-Bit großen Binaerzahl
//Das entspricht einem Wertebereich von -2^15..2^15-1
typedef struct
{
    unsigned b0 : 1;
    unsigned b1 : 1;
    unsigned b2 : 1;
    unsigned b3 : 1;
    unsigned b4 : 1;
    unsigned b5 : 1;
    unsigned b6 : 1;
    unsigned b7 : 1;

    unsigned b8 : 1;
    unsigned b9 : 1;
    unsigned b10 : 1;
    unsigned b11 : 1;
    unsigned b12 : 1;
    unsigned b13 : 1;
    unsigned b14 : 1;
    unsigned b15 : 1;
} binaerzahl;



/*------------------------------------------------------------------------*/
/* Die Funktion berechnet den Übertrag u(l), der bei der Addition +‘
von
 * Binaerzahlen erforderlich ist
 * Parameter: char uebertrag, der uebertrag u(l-1)
 *            char zahl1, Bit(l) der ersten Zahl
 *            char zahl2, Bit(l) der zweiten Zahl
```

```
 * Rueckgabewert: der Uebertrag u(l)
 */
/*-------------------------------------------------------------------------*/

char uebertrag(char uebertrag, char zahl1, char zahl2)
{
    //Uebertrag laut Definition im Skript
    //& ist binaere Multiplikation
    //^ ist Modulo-Addition, bzw. exclusive-or
    return (zahl1 & zahl2) + ((zahl1 ^ zahl2) & uebertrag);
}


/*-------------------------------------------------------------------------*/
/* Die Funktion addiert zwei Binaerzahlen
 * Dabei sind negative Zahlen im Zweierkomplement dargestellt
 * Parameter: binaerzahl zahl1, erste Zahl als Bitfeld
 *            binaerzahl zahl2, zweite Zahl als Bitfeld
 * Rueckgabewert: zahl1 + zahl2 als Bitfeld
 */
/*-------------------------------------------------------------------------*/

binaerzahl addieren(binaerzahl zahl1, binaerzahl zahl2)
{
    //Variablen fuer Rueckgabewert und Uebertrag
    binaerzahl rueck;
    char ueber;

    //Bit 1 berechnet sich aus der Modulo-Addition de jeweiligen Bits
    rueck.b0 = zahl1.b0 ^ zahl2.b0;
    //Uebertrag eins berechnet sich aus der Bit-Multiplikation der
    //jeweiligen Bits
    ueber = zahl1.b0 & zahl2.b0;
    //alle folgenden Bits berechnen sich aus der Modulo-Addition
    //der jeweiligen Bits und des vorigen Uebertrags
    rueck.b1 = zahl1.b1 ^ zahl2.b1 ^ ueber;
    //der Uebertrag berechnet sich in der Funktion uebertrag()
    ueber = uebertrag(ueber, zahl1.b1, zahl2.b1);
    rueck.b2 = zahl1.b2 ^ zahl2.b2 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b2, zahl2.b2);
    rueck.b3 = zahl1.b3 ^ zahl2.b3 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b3, zahl2.b3);
```

```
    rueck.b4 = zahl1.b4 ^ zahl2.b4 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b4, zahl2.b4);
    rueck.b5 = zahl1.b5 ^ zahl2.b5 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b5, zahl2.b5);
    rueck.b6 = zahl1.b6 ^ zahl2.b6 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b6, zahl2.b6);
    rueck.b7 = zahl1.b7 ^ zahl2.b7 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b7, zahl2.b7);

    rueck.b8 = zahl1.b8 ^ zahl2.b8 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b8, zahl2.b8);
    rueck.b9 = zahl1.b9 ^ zahl2.b9 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b9, zahl2.b9);
    rueck.b10 = zahl1.b10 ^ zahl2.b10 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b10, zahl2.b10);
    rueck.b11 = zahl1.b11 ^ zahl2.b11 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b11, zahl2.b11);
    rueck.b12 = zahl1.b12 ^ zahl2.b12 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b12, zahl2.b4);
    rueck.b13 = zahl1.b13 ^ zahl2.b13 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b13, zahl2.b13);
    rueck.b14 = zahl1.b14 ^ zahl2.b14 ^ ueber;
    ueber = uebertrag(ueber, zahl1.b14, zahl2.b14);
    rueck.b15 = zahl1.b15 ^ zahl2.b15 ^ ueber;

    //der Uebertrag des letzen Bits wird einfach vergessen
    //Rueckgabe des Summanden
    return rueck;
}


/*-------------------------------------------------------------------------*/
/* Funktion zur Bildung des Einerkomplements K1
 * Parameter: binaerzahl zahl, Zahl von der K1 gebildet werden soll
 * Rueckgabewert: das Einerkomplement von zahl
 */
/*-------------------------------------------------------------------------*/

binaerzahl einerkomplement(binaerzahl zahl)
{
    //Variable fuer den Rueckgabewert
    binaerzahl rueck;
```

```
    //jedes Bit des Einerkomplements wird durch Modulo-Addition mit
1
    //gebildet
    rueck.b0 = zahl.b0 ^ 0x1;
    rueck.b1 = zahl.b1 ^ 0x1;
    rueck.b2 = zahl.b2 ^ 0x1;
    rueck.b3 = zahl.b3 ^ 0x1;
    rueck.b4 = zahl.b4 ^ 0x1;
    rueck.b5 = zahl.b5 ^ 0x1;
    rueck.b6 = zahl.b6 ^ 0x1;
    rueck.b7 = zahl.b7 ^ 0x1;

    rueck.b8 = zahl.b8 ^ 0x1;
    rueck.b9 = zahl.b9 ^ 0x1;
    rueck.b10 = zahl.b10 ^ 0x1;
    rueck.b11 = zahl.b11 ^ 0x1;
    rueck.b12 = zahl.b12 ^ 0x1;
    rueck.b13 = zahl.b13 ^ 0x1;
    rueck.b14 = zahl.b14 ^ 0x1;
    rueck.b15 = zahl.b15 ^ 0x1;

    //Rueckgabe des Einerkomplements
    return rueck;
}


/*-------------------------------------------------------------------------*/
/* Funktion zur Bildung des Zweierkomplements K2
 * Parameter: binaerzahl zahl, zahl von de K2 gebildet werden soll
 * Rueckgabewert: Zweierkomplement von zahl
 */
/*-------------------------------------------------------------------------*/

binaerzahl zweierkomplement(binaerzahl zahl)
{
    //Variable fuer Rueckgabewert
    binaerzahl rueck;
    //Variable fuer Einerkomplement von zahl
    binaerzahl einer = einerkomplement(zahl);
    //Bitfeld fuer Zahl 1
    binaerzahl binaer1 = {1, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0};
```

```
    //Zweierkomplement berechnet sich aus dem Einerkomplement von Zahl
    //plus (modulo) 1
    rueck = addieren (einer, binaer1);

    //Rueckgabe des Zweierkomplements
    return rueck;
}


/*---------------------------------------------------------------------------*/
/* Funktion, die Bits 0 und 1 erzeugt
 * Parameter: short zahl, umzuwandelnde Zahl
 * Rueckgabewert: entsprechendes Bit
 */
/*---------------------------------------------------------------------------*/

char toBit(short zahl)
{
    //Variable fuer Ergebnis
    char ergebnis;
    //wenn zahl %  0 ist, dann ist das Bit 0, sonst 1
    if((zahl % 2) == 0)
        ergebnis = 0;
    else
        ergebnis = 1;

    //Rueckgabe des Bits
    return ergebnis;
}


/*---------------------------------------------------------------------------*/
/* Funtkion, die eine short-Zahl in ein Bitfeld umwandelt
 * Parameter: short uebergabezahl, umzuwandelnde Zahl
 * Rueckgabewert: Bitfeld fuer uebergabezahl
 */
/*---------------------------------------------------------------------------*/

binaerzahl toBitfeld(short uebergabezahl)
{
    //Variable fuer Bitfeld
```

```
binaerzahl bzahl;
//Variable, um Zahl fuer die Umwandlung zu merken
short zahl = uebergabezahl;

//jeweiliges Bit merken und zahl aktualisieren,
//bis uebergabezahl komplett umgewandelt ist
bzahl.b0 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b1 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b2 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b3 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b4 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b5 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b6 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b7 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;

bzahl.b8 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b9 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b10 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b11 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b12 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b13 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b14 = toBit(zahl);
zahl = (zahl - zahl % 2) / 2;
bzahl.b15 = toBit(zahl);

//wenn es sich um eine negative Zahl handelt, muss sie ins das
//Zweierkomplment umgewandelt werden
if(uebergabezahl < 0)
```

```
        bzahl = zweierkomplement(bzahl);

    //Rueckgabe des Bitfeldes
    return bzahl;
}


/*--------------------------------------------------------------------------*/
/* Funktion, um ein Bitfeld auszugeben
 * Parameter: binaerzahl zahl, auszugebende zahl
 * Rueckgabewert: -
 *(
/*--------------------------------------------------------------------------*/

void printBinaer(binaerzahl zahl)
{
    //Bits nacheinander ausgeben
    printf("%1d ", zahl.b15);
    printf("%1d", zahl.b14);
    printf("%1d", zahl.b13);
    printf("%1d", zahl.b12);
    printf("%1d", zahl.b11);
    printf("%1d", zahl.b10);
    printf("%1d", zahl.b9);
    printf("%1d", zahl.b8);

    printf("%1d", zahl.b7);
    printf("%1d", zahl.b6);
    printf("%1d", zahl.b5);
    printf("%1d", zahl.b4);
    printf("%1d", zahl.b3);
    printf("%1d", zahl.b2);
    printf("%1d", zahl.b1);
    printf("%1d", zahl.b0);
}


/*--------------------------------------------------------------------------*/
/* Funktion zum Subtrahieren von Binaerzahlen,
 * dabei muessen negative Zahlen im Zweierkomplement stehen
 * Parameter: binaerzahl zahl1, erste Zahl
 *            binaerzahl zahl2, abzuziehende Zahl
```

```
 * Rueckgabewert: zahl1 - zahl2
 */
/*--------------------------------------------------------------------------*/

binaerzahl subtrahieren(binaerzahl zahl1, binaerzahl zahl2)
{
    //Variable fuer Rueckgabewert
    binaerzahl rueck;

    //Binare Subtraktion ist Addition des Zweierkomplements
    rueck = addieren(zahl1, zweierkomplement(zahl2));

    return rueck;
}


/*--------------------------------------------------------------------------*/
/* Funktion zur Bildung der Zweierpotent 2^n
 * Parameter: short n, die Potenz
 * Rueckgabewert: 2^n
 */
/*--------------------------------------------------------------------------*/

short zweierpotenz(short n)
{
    //Variablen fuer das Ergebnis und Zaehler
    short ergebnis = 1;
    short i;

    //Zweierpotenz berechnen
    for (i = 1; i <= n; i++)
        ergebnis *= 2;

    //Ergebnis zurueckgeben
    return ergebnis;
}


/*--------------------------------------------------------------------------*/
/* Funktion, die Bitfeld wieder in short-Zahl umwandelt
 * Parameter: binaerzahl zahl, umzuwandelndes Bitfeld
 * Rueckgabewert: short-Zahl
```

```
 */
/*-------------------------------------------------------------------------*/

short toShort(binaerzahl zahl)
{
    //Variable fuer das Ergebnis
    short ergebnis = 0;
    //Merker fuer negative Zahlen
    char negativ = 0;

    //wenn das letzte Bit 1 ist, ist die Zahl negativ
    if (zahl.b15 == 1)
    {
        //in diesem Fall, merken, dass negativ und Zweierkomplement
bilden
        negativ = 1;
        zahl = zweierkomplement(zahl);
    }

    //Wert des Bitfeldes aufsummieren
    //Ergebnis ist Summe von 0 bis n von b(n)*2^n
    ergebnis = zahl.b0 * zweierpotenz(0);
    ergebnis += zahl.b1 * zweierpotenz(1);
    ergebnis += zahl.b2 * zweierpotenz(2);
    ergebnis += zahl.b3 * zweierpotenz(3);
    ergebnis += zahl.b4 * zweierpotenz(4);
    ergebnis += zahl.b5 * zweierpotenz(5);
    ergebnis += zahl.b6 * zweierpotenz(6);
    ergebnis += zahl.b7 * zweierpotenz(7);

    ergebnis += zahl.b8 * zweierpotenz(8);
    ergebnis += zahl.b9 * zweierpotenz(9);
    ergebnis += zahl.b10 * zweierpotenz(10);
    ergebnis += zahl.b11 * zweierpotenz(11);
    ergebnis += zahl.b12 * zweierpotenz(12);
    ergebnis += zahl.b13 * zweierpotenz(13);
    ergebnis += zahl.b14 * zweierpotenz(14);
    ergebnis += zahl.b15 * zweierpotenz(15);

    //bei einem negativen Wert, Ergebnis negativ machen
    if (negativ == 1)
        ergebnis = -ergebnis;
```

```
    //Ergebnis zurueckgeben
    return ergebnis;
}


/*------------------------------------------------------------------------*/
int main()
{
    //Variablen fuer Zahlen und Loesungen
    short eingabe1, eingabe2, loesung;
    binaerzahl beingabe1, beingabe2, bloesung;

    //Zwei Zahlen eingeben und einlesen
    printf("Geben Sie eine Integer-Zahl ein ([-32768, 32767]): ");
    scanf("%hd", &eingabe1);
    printf("Geben sie eine zweite Integer-Zahl ein([-32768, 32766]):
");
    scanf("%hd", &eingabe2);

    //Zahlen in Binaerzahlen umwandeln und Ergebnis binaer berechnen
    beingabe1 = toBitfeld(eingabe1);
    beingabe2 = toBitfeld(eingabe2);
    bloesung = addieren(beingabe1, beingabe2);

    //Berechnung als Bitfeld ausgeben (zahl1 + zahl2 = loesung)
    printBinaer(beingabe1);
    printf(" + ");
    printBinaer(beingabe2);
    printf(" = ");
    printBinaer(bloesung);
    printf("\n");

    //die Loesung wieder in eine Zahl umwandeln und zur Kontrolle
    //ausgeben
    loesung = toShort(bloesung);
    printf("%hd + %hd = %hd\n", eingabe1, eingabe2, loesung);

    //jetzt subtrahieren
    bloesung = subtrahieren(beingabe1, beingabe2);

    //Berechnung als Bitfeld ausgeben (zahl1 - zahl2 = loesung)
    printBinaer(beingabe1);
    printf(" - ");
```

```
    printBinaer(beingabe2);
    printf(" = ");
    printBinaer(bloesung);
    printf("\n");

    //die Loesung wieder in eine Zahl umwandeln und zur Kontrolle
    //ausgeben
    loesung = toShort(bloesung);
    printf("%hd - %hd = %hd\n\n", eingabe1, eingabe2, loesung);
}
```