Musterlösung

Semesterendprüfung INCO 15.01.2014 Maximum: 36 Punkte

1. Arithmetik, Zahlensysteme und Boole'sche Funktionen

1. Teilaufgabe

(a) Dezimal: 170 Hexadezimal: AA

(b) Minuend: 72546; Subtrahend: 03809;
Nun zuerst das 10-er Komplement des Subtrahenden: 96191
Dann die Addition mit Übertrag: 72546
96191
168737
Ersatz Übertrag: 68737
Resultat: **68737**

(c) Die Darstellung ganzer Zahlen erfordert auch die Darstellung negativer Zahlen. Da zum Beispiel das duale Zahlensystem kein negatives Vorzeichen kennt, muss man auf ein Hilfsmittel zurückgreifen. Wenn man das MSB dazu benutzt, positive und negative Zahlen darzustellen, hat die Null zwei Darstellungen. Um dies zu vermeiden wird die Komplement-Arithmetik verwendet.

(d) Zuerst die 1-er-Komplement Zahl berechnen: 11011000
Zahl selbst: $00100111_2 = 39_{10}$
also: $11011001_2 = -39_{10}$

(e) Zuerst $+57823_{10}$ binär dargestellt:
1110 0001 1101 1111; nun: MSB = 1 heisst: Negative Zahl.
Also muss 57823_{10} wie folgt dargestellt sein im Binärsystem:
0000 0000 0000 0000 1110 0001 1101 1111
Dann: Einer-Komplement davon:
1111 1111 1111 1111 0001 1110 0010 0000_2
Zweier-Komplement, also Resultat:
1111 1111 1111 1111 0001 1110 0010 0001_2 $= -57823_{10}$

2. Teilaufgabe

(a) *Gewichteter Code*: Falls die Dezimalzahlen eindeutig aufgrund eines fest vorgegebenen Polynoms und seinen n Koeffizienten dargestellt werden. **Beispiel: BCD-Code**
Komplementierender Code: Falls das Codewort des 9er-Komplements der Dezimalzahl das 1er-Komplement des Codeworts ist, das die Dezimalzahl codiert. **Beispiel: Excess-3-Code**.

(b) Zuerst Umwandlung in eine Dezimalzahl:
Dezimal: $2^6+2^5+2^2+2^1+2^0+0.5 = 64+32+4+2+1+0.5 = 103.5$
BCD: 0001 0000 0011 **.** 0101

(c) ASCII-Code für ZHAW: 1011010 1001000 1000001 1010111

3. Teilaufgabe

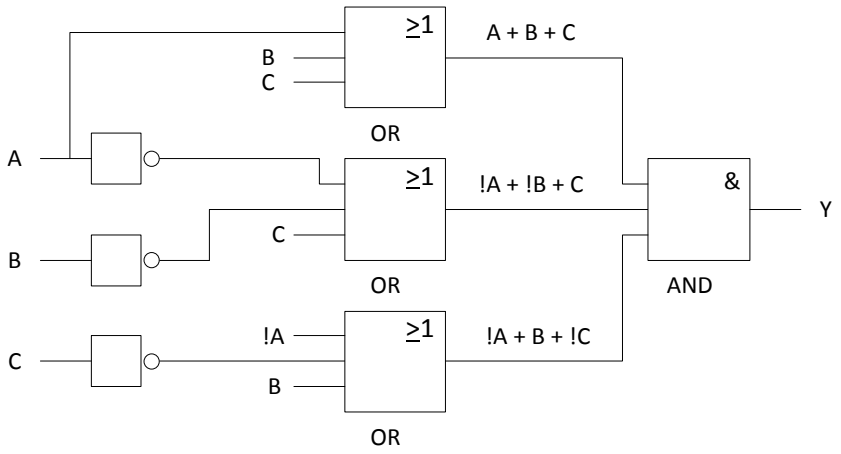
(a) Produkte von Summen: Hierzu müssen **die Nullen** des Output der Wahrheitstabelle betrachtet werden. Danach mit der Formel von **de Morgan** umformen: (Punkt entspricht: AND; + entspricht: ODER)

Also: !Y = (!A·!B·!C) + (A·B·!C) + (A·!B·C); nun **de Morgan**:
Y = (A+ B+C) · (!A+!B+C) · (!A+ B+!C)
Das ist nun ein Produkt von Summen.

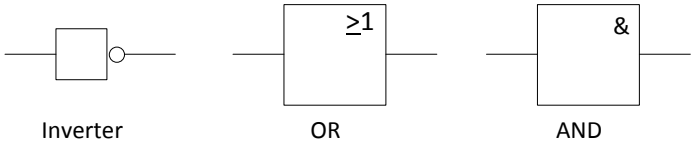
(b) Logische Schaltung:

Es soll die folgende Boole'sche Funktion in einer Schaltung abgebildet sein:

Y = (A + B + C) & (!A + !B + C) & (!A + B + !C)



Legende für die Gates:



Schreibweise für Inversion:
!X = (X nicht) = ¯X

Schreibweise für OR: +

Schreibweise für AND: &

(c) Wiederum die Regel von de Morgan anwenden:
f(A,B,C) = (A·B·C) + (!A·!B·C) = Resultat
auch möglich: f(A,B,C) = C · !(A **EXOR** B) = Resultat

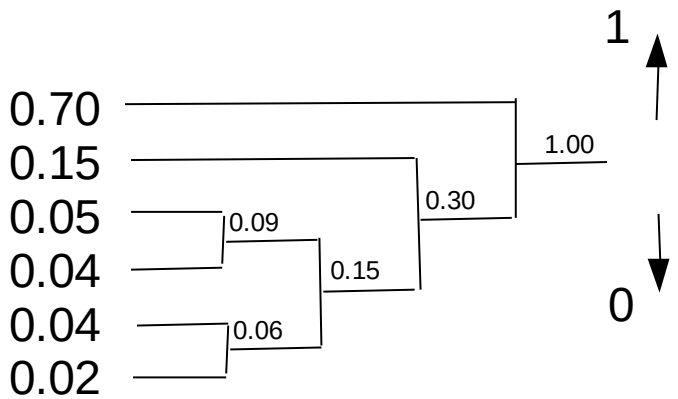
2. Entropie

```
a=[0.7  0.15  0.05  0.04  0.04  0.02];
I=log(1./a)/log(2);
H=sum(a.*log(1./a)/log(2))
R=4-H;
```

a) **I = 4.3219 Bit**
b) H = 1.4712 Bit/Symbol
c) R = 2.5288 Bit/Symbol
d) Die Entropie nimmt zu.
Begründung: Aus der Quelle kommen viel mehr Nullen als Einsen, da das erste Symbol 0000b sehr viel häufiger vorkommt als alle anderen. Durch den BSC werden daher mehr Nullen in Einsen verfälscht als umgekehrt. Das heisst, dass Nullen und Einsen am Ausgang des BSC ausgeglichener sind als am Eingang.

3. Huffman

a)



Code:

0000b	P(0000b) = 0.70	**1**
0001b	P(0001b) = 0.15	**01**
0010b	P(0010b) = 0.05	**0011**
0100b	P(0100b) = 0.04	**0010**
1000b	P(1000b) = 0.04	**0001**
1111b	P(1111b) = 0.02	**0000**

b)
b=[1 2 4 4 4 4];
L=a*b';
Mittlere CW-Länge : **E[L]= 1.6000 Bit/Symbol**

c) Da L >H ist der Huffman Code nicht ideal. Man könnte einen besseren Huffman Code mit noch kleinerer Redundanz finden, indem man alle möglichen **Doppelsymbole** bildet (total 6*6=36 Stück, z. B. {0000b 0100b}) und mit diesen einen Huffman Baum, resp. Huffman Code entwickeln würde.

d) Nein, die Huffman Codierung ist ein **Quellen**codierungsverfahren, das Redundanz entfernen soll. Im Gegensatz dazu wir bei der Kanalcodierung gezielt Redundanz hinzugefügt, um Fehler bei der Übertragung erkennen und evtl. korrigieren zu können.

4. Lempel-Ziv-Codierung

G'E'R'B'ER 'GERBEN' W'ERBER W'ERBEN E'RBEN W'A'RT'EN.

Grösse des Vorschaubuffers: L = 15
Grösse des Suchbuffers: S = 15

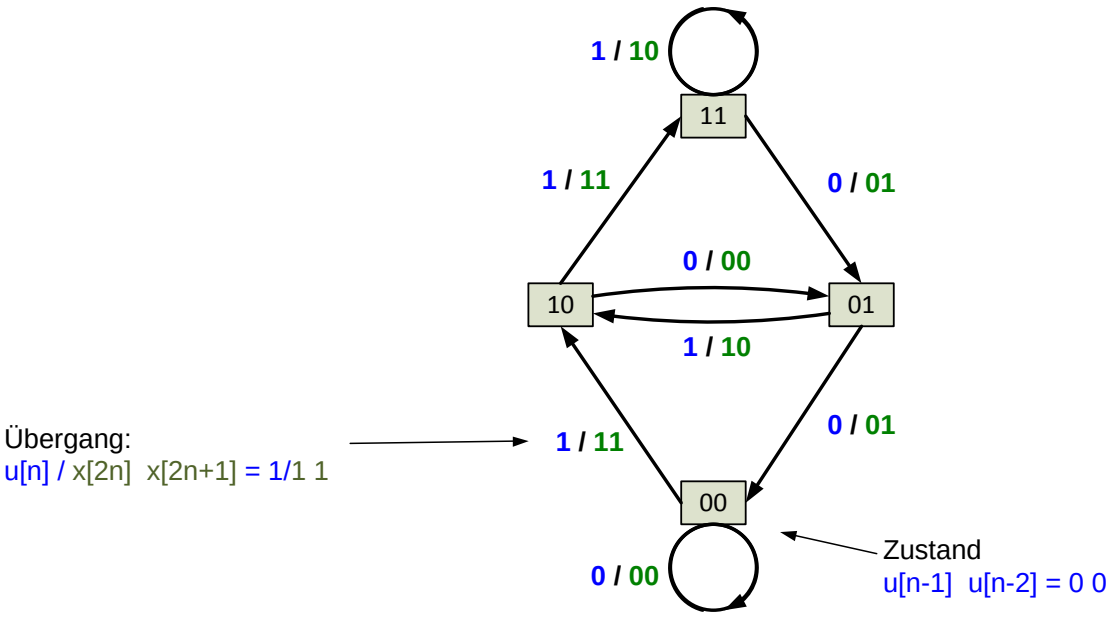
a) (0,0,G) (0,0,E) (0,0,R) (0,0,B) (3,2,_) (7,5,N) (7,1,W) (14,6,W) (14,6,E) (6,5,W) (0,0,A) (7,1,T) (7,2,.)

b) Es sind 13 Token mit je (4 + 4 + 8) Bit, also total 26 Bytes.
Ursprünglich: 41 Bytes. Ja, Kompression findet statt.
Kompressionsrate: R = 26/41 = 0.634

c) Für den Vorschaubuffer L = 31: ➔ 5 Bits erforderlich. Für den Suchbuffer S = 2047: ➔ 11 Bits erforderlich. Für die Angabe des nächstfolgenden Zeichens: 8 Bits erforderlich. Total: 24 Bits = 3 Byte grosse Token erforderlich.

5. Faltungscode

a) Das vollständige Zustandsdiagramm ist wie folgt:



b) Das Codewort zum Infowort $\underline{u}$ = [110100] lautet:
CW = [11 11 01 10 00 01]

c) d_{free} beträgt 3