Lösungen zum 4. Übungsblatt Informatik II (SS 96)

Jan Niklas Fingerle (zwenti@fsinfo.cs.uni-sb.de)
Michael Mehl (mehl@dfki.uni-sb.de)

Lösung zu Aufgabe 4.1 [Zahlendarstellung I]

1.

	Dezimal	Binär	Hexadezimal	Octal
	308	100110100	134	464
+	2938	101101111010	B7a	5572
=	3246	110010101110	CAE	6256
+	44576	1010111000100000	AE20	127040
=	47822	1011101011001110	BACE	135316

2. $twoc_9(-300) = 1011010100_2$

   $twoc_9(12) = 0000001100_2$

3. $twoc_9(300) = inv(twoc_9(-300)) + 1 = 0100101011_2 + 1_2 = 0100101100_2$

   $twoc_9(-12) = inv(twoc_9(12)) + 1 = 0000001100_2 + 1_2 = 1111110100_2$

4.

Bitfolge	Zweierkomplement	Darst. mit Vorzeichenbit
1100	-4	-4
0010	+2	+2

Lösung zu Aufgabe 4.2 [Zahlendarstellung II]

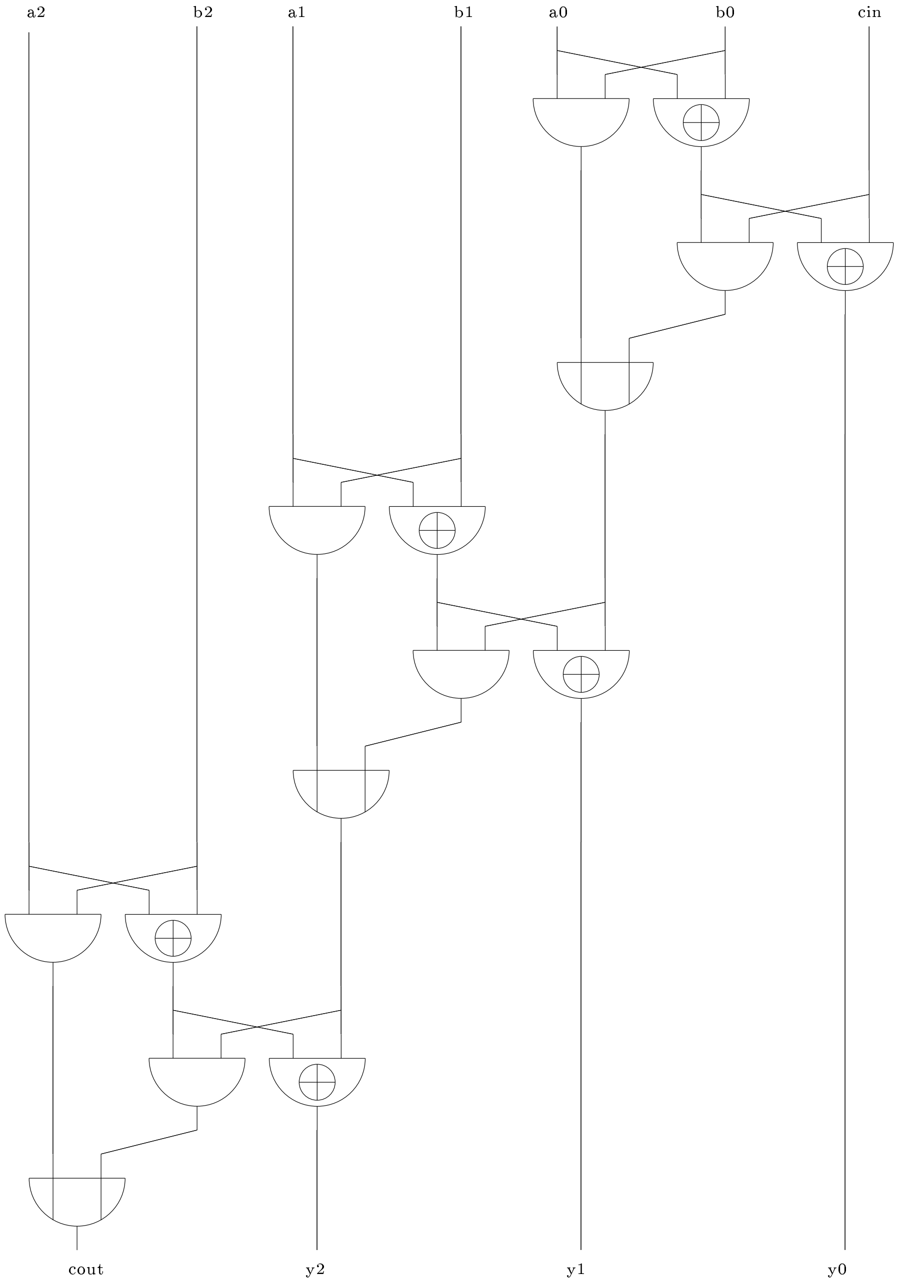
```
;; von Florian Mueller

(define (bin n s)
  (cond ((= s 0) (list n))
        ((>= n (expt 2 s)) (cons 1 (bin (- n (expt 2 s)) (- s 1))))
        (else (cons 0 (bin n (- s 1))))))

(define (twoc n m)
  (cond ((or (>= m (expt 2 n)) (< m (- 0 (expt 2 n)))) 'error)
        ((> 0 m) (cons 1 (bin (+ (expt 2 n) m) (- n 1))))
        (else (cons 0 (bin m (- n 1))))))

(define (bin-inv m l)
  (cond ((= 0 l) 0)
        (else (+ (* (car m) (expt 2 (- l 1))) (bin-inv (cdr m) (- l 1))))))

(define (twoc-inv m)
  (cond ((not (list? m)) 'error)
        ((< (length m) 2) 'error)
        ((not (or (= (car m) 1) (= (car m) 0))) 'error)
        (else (+ (* (expt 2 (- (length m) 1)) (- 0 (car m))) (bin-inv (cdr m)
          (length (cdr m)))))))
```

Lösung zu Aufgabe 4.3 [Zahlendarstellung III]
Beweis durch Einsetzung der Definition:

$$\begin{aligned}
& [a_{n-1}, ..., a_l]_B \cdot B^l + [a_{l-1}, ..., a_0]_B \\
= \quad & (\sum_{i=0}^{n-1-l} a_{l+i} B^i) B^l + \sum_{i=0}^{l-1} a_i B^i \\
= \quad & \sum_{i=0}^{n-1-l} a_{l+1} B^{l+i} + \sum_{i=0}^{l-1} a_i B^i \\
= \quad & \sum_{i=l}^{n-1} a_i B^i + \sum_{i=0}^{l-1} a_i B^i \\
= \quad & \sum_{i=0}^{n-1} a_i B^i \\
= \quad & [a_{n-1}, ..., a_0]_B
\end{aligned}$$

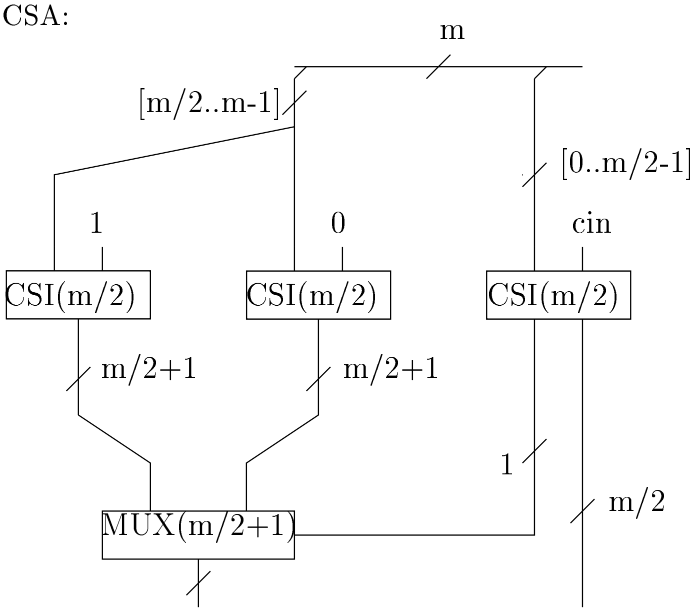
Lösung zu Aufgabe 4.4 [Carry-Chain Addierer]

3-Bit Carry-Chain-Addierer mit Carryeingabe



Lösung zu Aufgabe 4.5 [Incrementer]

1. Ich konstruiere zunächst einen Addierer für a+0+Übertrag (und nenne ihn CSI).
   (a) Als 1-Bit-CSI nehme ich einen Halbaddierer.
   (b) Nun kann ich rekursiv einen $m = 2^n$-Bit-CSI konstruieren nach dem Schema des CSA:



   Einen m-Bit-CSI erhalte ich nun aus einem m-Bit-Inkrementierer, indem ich den carry-in auf 1 setze.

2. Wertetabelle HA/FA:

b_n	a_n	c_{in}	HA	FA
0	0	0	(0;0)	(0;0)
0	0	1	(0;1)	(0;1)
0	1	0	(0;1)	(0;1)
0	1	1	(1;0)	(1;0)
1	0	0	(0;0)	(0;1)
1	0	1	(0;1)	(1;0)
1	1	0	(0;1)	(1;0)
1	1	1	(1;0)	(1;1)

   Die Konstruktion entspricht einem CSA, bei dem aber die Volladdierer durch Halbaddierer ersetzt worden sind und die Eingänge für b gestrichen worden sind. Nach Wertetabelle entspricht dies wiederum also einem CSA, bei dem die Eingänge b_n auf 0 gesetzt sind, da die HA/FA diese Eigenschaft besitzen und jeder 1-Bit-Addierer in einer CS-Konstruktion genau die Eingaben a_n, b_n und ein carry für ein bestimmtes n haben. Ein CSA berechnet, wie in der Vorlesung gezeigt $[a]_{bin} + [b]_{bin} + carry$. Also berechnet der oben konstruierte CSI $[a]_{bin} + 0 + carry = [a]_{bin} + carry$. Da wir carry auf 1 setzen ist der konstruierte CSI also auch wirklich ein Incrementer.

   Tiefe folgt aus dem Beweis über CSA in der Vorlesung.

Lösung zu Aufgabe 4.6 [Was ist das?]

1. Es wird die Addition (modulo B^n) in der Schulmethode beschrieben. Dabei sind x und y die beiden Summanden, die c_i sind die Überträge, $[c_{n-1}, s_{n-1}, ..., s_0]_B$ ($[s_{n-1}, ..., s_0]_B$) ist das Ergebnis.

2. Beweis per Induktion über n:

   n=1: Nach Definition $s_0 = (x_0 + y_0 + 0) \bmod B$. Also gilt die Behauptung 2 für n=1.

   n → n+1: Nach Aufgabe 4.3 und Induktionsvoraussetzung gilt:

$$\begin{aligned} & [s_n, ..., s_0]_B \\ = \; & s_n \cdot B^n + [s_{n-1}, ..., s_0] \\ = \; & (x_n + y_n + c_{n-1}) \cdot B^n + ([x_{n-1}, ..., x_0] + [y_{n-1}, ..., y_0] \bmod B^n) \bmod B^{n+1} \end{aligned}$$

   Da der carry gerade genau dann erzeugt wird, wenn durch das „modB^n“ B^n abgezogen wird (da die Summanden in der Klammer kleiner sind als B^n ist das Ergebnis immer kleiner als $2B^n$, daher wird durch „ mod B^n“ höchstens B^n abgezogen), ergibt sich die Behauptung 2

$$\begin{aligned} & (x_n + y_n + c_{n-1}) \cdot B^n + ([x_{n-1}, ..., x_0] + [y_{n-1}, ..., y_0] \bmod B^n) \bmod B^{n+1} \\ = \; & (x_n + y_n) \cdot B^n + c_{n-1} \cdot B^n + ([x_{n-1}, ..., x_0] + [y_{n-1}, ..., y_0] \bmod B^n) \bmod B^{n+1} \\ = \; & ((x_n + y_n) \cdot B^n + [x_{n-1}, ..., x_0] + [y_{n-1}, ..., y_0]) \bmod B^{n+1} \\ = \; & ([x_n, ..., x_0] + [y_n, ..., y_0]) \bmod B^{n+1} \end{aligned}$$

   nach Aufgabe 4.3. Wiederum da der carry gerade genau dann erzeugt wird, wenn durch das „ mod B^n“ B^n abgezogen wird, folgt daraus nach Aufgabe 4.3 Behauptung 1.

Lösung zu Aufgabe 4.7 [BCD-Zahlen]

1. $f_3(8) = 000000001000_{BCD}$
   $f_3(981) = 100110000001_{BCD}$

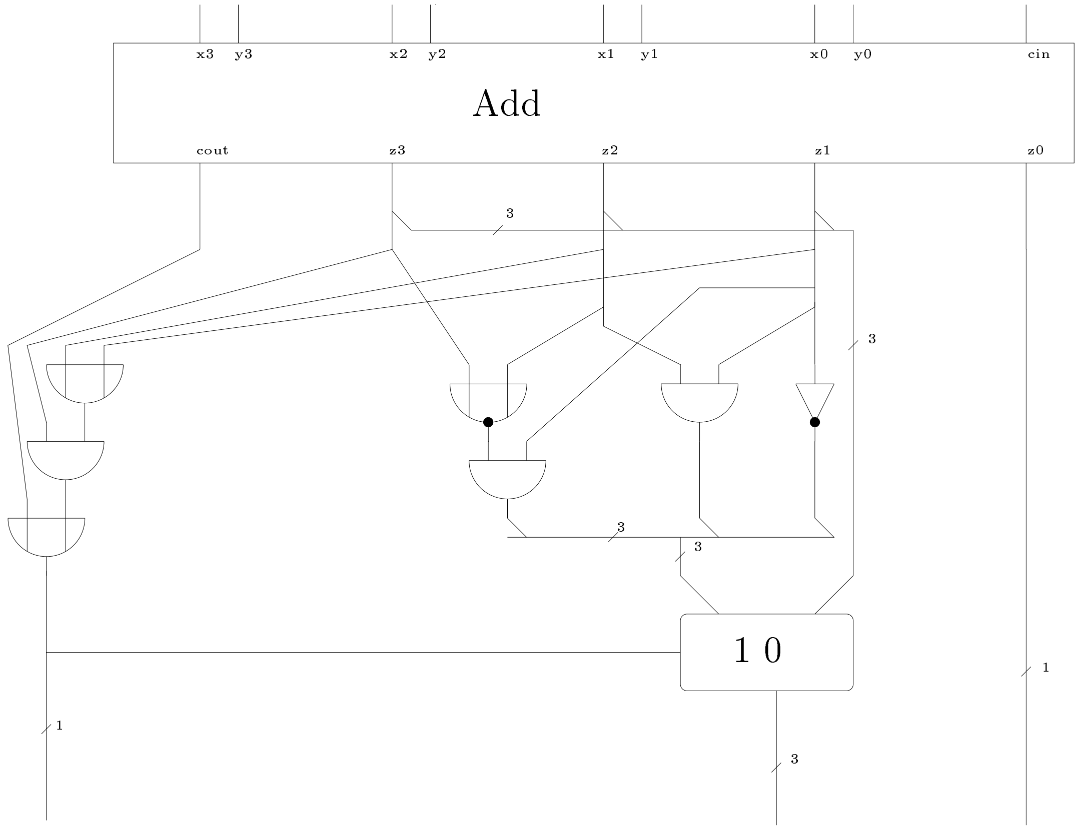
2. Ich konstruiere zunächst einen 1-Ziffer-BCD-Addierer:

   (a) Addiere zunächst die $[x_3, ..., x_0]_{BCD}$ und $[y_3, ..., y_0]_{BCD}$, sowie den carry-in binär. Das Ergebnis $[c_{out}, z_3, ..., z_0]$ ist in jedem Fall kleiner als 20.

   (b) Nun prüfen wir ob das Ergebnis größer als 10 ist mittels einer Schaltung, die $sc_{out} = sc_{out} \vee (s_3 \wedge (s_2 \vee s_1))$ berechnet. sc_{out} ist, wie leicht nachzuprüfen ist, genau dann 1, wenn das Ergebnis grösser als 9 ist. Damit ist es unser carry-out.

   (c) Falls das Ergebnis kleiner als 9 ist, so ist $(s_3, ..., s_0)$ schon die Darstellung der entsprechenden BCD-Ziffer. Ist es grösser, so reichen wir 1 als Übertrag weiter, müssen also an dieser Stelle 10 vom ursprünglichen Ergebnis subtrahieren. Hierzu können wir einen weiteren Addierer benutzen, der $[0110]_{bin}$ addiert, wenn $sc_{out} = 1$, sonst $[0000]_{bin}$ addiert. Also addiert er $[0, sc_{out}, sc_{out}, 0]_{bin}$.

   (d) Ersatzweise können wir eine Schaltung bauen, die nur die Differenz $[s_3, ..., s_0]_{bin}$ - 10 berechnet und dann das richtige Ergebnis mit einem MUX auswählen. Diese Variante wird in der Zeichnung dargestellt:



Die so konstruierten 1-Ziffer-BCD-Addierer können nun als Carry-Lookahead-Addierer, Carry-Chain-Addierer oder Conditional-Sum-BCD-Addierer zu einem n-Ziffern-BCD-Addierer zusammenschalten.