RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

2. Klausur zur Diplom–Vorprüfung (DPO 89) in „Rechnerstrukturen“

23. März 1999

Aufgabe 1: (17 Punkte)

(a) Gegeben sei die folgende Boolesche Funktion f:

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$f(x_1, \ldots, x_4)$	1	1	0	1	0	0	0	0	1	1	0	1	1	1	0	1

(i) Wenden Sie das Karnaugh-Verfahren auf die DNF von f an, um eine möglichst kurze disjunktive Form von f zu erhalten.

(ii) Zeigen Sie, daß die in (i) gefundene Darstellung $\mathrm{DF}_{\min}$ in folgendem Sinne nicht optimal ist:
Es existiert eine Boolesche Formel, die f beschreibt und weniger zweistellige Operatoren benötigt als $\mathrm{DF}_{\min}$.

(b) Gegeben sei die folgende Boolesche Funktion f:

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$f(x_1, \ldots, x_4)$	0	0	1	1	0	1	0	1	1	0	1	0	1	1	0	0

(i) Bestimmen Sie alle Primimplikanten von f.

(ii) Bestimmen Sie zwei unterschiedliche disjunktive Formen (DF) von f, die beide das Ergebnis des Quine-McCluskey-Verfahrens sein können.

Aufgabe 2: (16 Punkte)

(a) Geben Sie das Schaltbild eines 4-Bit-Serien-Addierwerks an und beschreiben Sie dessen Funktionsweise.

(b) Beschreiben Sie das 4-Bit-Serien-Addierwerk durch einen Mealy-Automaten.

Aufgabe 3: (17 Punkte)

Schreiben sie ein gut kommentiertes Assembler-Programm, welches das folgende Programm — den Euklidischen Algorithmus zur Bestimmung des größten gemeinsamen Teilers — umsetzt. Sie dürfen dabei jeden der in der Vorlesung vorgestellten Assembler benutzen.

```
Funktion ggt(a,b);

Eingabe: a,b
Ausgabe: ggt(a,b)

repeat
r := a mod b;
f := a div b;
a := b;
b := r;
until r = 0;
ggt(a,b) := a;
```

Beachten Sie:

- Sie dürfen nicht die Operation "mod" direkt in Assembler ausführen.
- Beim Teilen zweier Zahlen erhalten Sie nur den ganzzahligen Anteil.
- Gehen Sie von folgender Speicherbelegung aus:

a	steht in	M(1),	
b	steht in	M(2),	
ggt(a,b)	soll in	M(0)	abgelegt werden.

- Die Inhalte von M(1) und M(2) dürfen nicht verändert werden.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Testklausur zur Vorlesung „Rechnerstrukturen“

10. August 1998

Aufgabe 1: (10 Punkte)

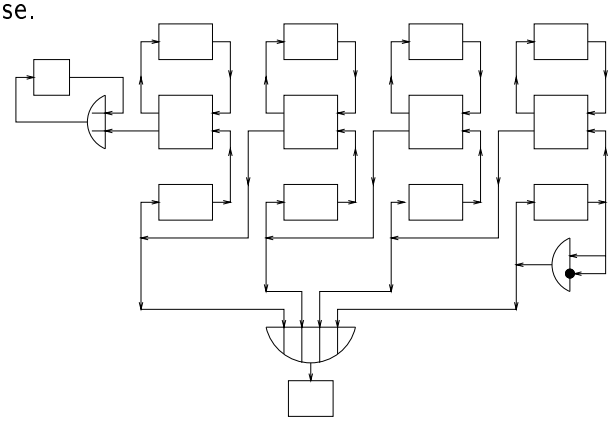
Beweisen Sie, daß die folgenden Mengen funktional vollständig sind:

a: $\{\vee, \wedge, \neg\}$, **b:** $\{\vee, \neg\}$, **c:** $\{\oplus, \wedge\}$, **d:** $\{\uparrow\}$, **e:** $\{\downarrow\}$.

Verwenden Sie nur den Darstellungssatz für Boolesche Funktionen, d.h. Zwischenschritte sind zu beweisen.

Aufgabe 2: (10 Punkte)

a: Vervollständigen Sie das folgende Schema eines 4-Bit-von Neumann-Addierwerks und beschreiben Sie dessen Funktionsweise.



b: Wieviele Takte braucht ein n-Bit-von Neumann-Addierwerk im schlechtesten Fall? Begründen sie Ihre Antwort.

c: Beschreiben Sie ein n-Bit-von Neumann-Addierwerk als Mealy-Automat für ein beliebiges $n \in \mathbb{N}$.

Aufgabe 3: (10 Punkte)

a: Definieren Sie die Aufgabe eines d-Demultiplexers ($d-$DeMUX) für ein beliebiges $d \in \mathbb{N}$.

b: Entwerfen Sie einen 3-DeMUX.

c: Benutzen Sie Ihren 3-DeMUX als Baustein für ein Schaltnetz zur Berechnung der folgenden Funktion:

$$f(y_1, y_2, y_3) = \overline{y}_1 y_2 y_3 \vee y_1 \overline{y}_2 y_3 \vee \overline{y}_1 \overline{y}_2 \overline{y}_3.$$

Aufgabe 4: (10 Punkte)

Schreiben Sie ein gut kommentiertes Assemblerprogramm im orthogonalen Assembler zur Lösung der folgenden Aufgabe:

Im Speicher steht eine Matrix M der Größe $n \times n$ mit Einträgen aus $\mathbb{N}$. Bestimmen Sie den Index der lexikographisch kleinsten Zeile. D.h. es ist $x \in \{1, ..., n\}$ zu bestimmen mit folgender Eigenschaft:

$\forall i \in \{1, ..., n\} : \exists k \in \{1, ..., n+1\}$ mit:

$$\forall j \in \{1, ..., k-1\} : m_{x,j} = m_{i,j} \quad \text{und} \quad m_{x,k} < m_{i,k} \vee k = n+1.$$

Die Speicherbelegung ist wie folgt gegeben:

$$n = \delta(M_0) \quad \text{und} \quad m_{i,j} = \delta(M_{j+n\cdot(i-1)}) \text{ für } 1 \leq i, j \leq n.$$

Der Index der lexikographisch kleinsten Zeile soll am Ende im Register 0 liegen.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

27. April 1998

Musterlösungen zu Blatt 1

Lösung zu Aufgabe 1

Sei $b \in \mathbb{N}$ mit $b > 0$.
Behauptung: Dann ist jede natürliche Zahl z mit $0 \leq z \leq b^n - 1$ ($n \in \mathbb{N}$) eindeutig als Wort der Länge n über dem Alphabet Σ_b darstellbar durch

$$z = \sum_{i=0}^{n-1} z_i \cdot b^i$$

mit $z_i \in \Sigma_b$ für $i \in \{0, 1, \ldots, n-1\}$.

Beweis:
Zur Existenz der *b*-adischen Darstellung:
Gegeben sei eine natürliche Zahl $0 \leq x_0 \leq b^n - 1$.
Konstruktives Verfahren zur Bestimmung der Darstellung der Zahl x_0 zur Basis b:
Dividiere die gegebene Zahl x_0 mit Rest durch die Basis b. Führe dies so lange fort, bis das Divisionsergebnis x_j Null wird:

$$\begin{aligned} x_0 &\ /b = x_1 \text{ Rest } z_0 \\ x_1 &\ /b = x_2 \text{ Rest } z_1 \\ x_2 &\ /b = x_3 \text{ Rest } z_2 \\ &\ \vdots \\ x_{n-1} &\ /b = x_n \text{ Rest } z_{n-1} \end{aligned}$$

Da $x_0 \leq b^n - 1$ gilt, terminiert das Verfahren nach höchstens n Schritten und liefert die Reste $z_{n-1}, \ldots, z_0$.
Zeige: $x_0 = (z_{n-1} \ldots z_0)_b$.
Beweis dazu: Es gilt:

$$\begin{aligned} x_0 &= x_1 b + z_0 \\ &= (x_2 b + z_1) b + z_0 && = x_2 b^2 + z_1 b + z_0 \\ &\ \vdots \\ &= (\ldots (\underbrace{x_n b}_{=0} + z_{n-1}) b \ldots) b + z_0 && = z_{n-1} b^{n-1} + z_{n-2} b^{n-2} + \cdots + z_0 \\ &= \sum_{i=0}^{n-1} z_i b^i \\ &= (z_{n-1} \ldots z_0)_b \end{aligned}$$

Zur Eindeutigkeit der b-adischen Darstellung:
Wir benötigen zunächst die folgende Hilfsbehauptung:

$$\sum_{i=0}^{n-1}(b-1)\cdot b^i < b^n \tag{1}$$

Beweis von (1) mit vollständiger Induktion über n:
$\boldsymbol{n=1}$: Es gilt: $\sum_{i=0}^{0}(b-1)\cdot b^i = b^0 = 1 < b^1 = b$.
$\boldsymbol{n \to n+1}$: Es gilt:

$$\begin{aligned}\sum_{i=0}^{n}(b-1)\cdot b^i &= (b-1)\cdot b^n + \sum_{i=0}^{n-1}(b-1)\cdot b^i \\ &< (b-1)\cdot b^n + b^n \qquad \text{(nach Induktionsvoraussetzung)} \\ &= b^{n+1}\end{aligned}$$

Wir beweisen nun die Eindeutigkeit der b-adischen Darstellung mit Worten der Länge n einem Widerspruchsbeweis:
Annahme: Es existiert eine Zahl x mit $0 \le x \le b^n - 1$, die zwei verschiedene b-adischen Darstellungen $x = (z_{n-1}\dots z_0)_b = (z'_{n-1}\dots z'_0)_b$ besitzt. Sei x die kleinste Zahl mit dieser Eigenschaft.
Sei o.B.d.A. $z_{n-1} \neq 0$. [Falls $z_{n-1} = z'_{n-1} = 0$ gilt, so kann man die b-adische Darstellung von z mit Worten der Länge $n-1$ betrachten.] Dann gilt

$$\sum_{i=0}^{n-1} z_i \cdot b^i = \sum_{i=0}^{n-1} z'_i \cdot b^i. \tag{2}$$

Da x die kleinste Zahl mit zwei verschiedenen b-adischen Darstellungen ist, muß $z_{n-1} \neq z'_{n-1}$ gelten, da sonst auch die Zahl $x - z_{n-1}\cdot b^{n-1} = (z_{n-2}\dots z_0) = (z'_{n-2}\dots z'_0)$ zwei verschiedene b-adische Darstellungen hätte.
Sei o.B.d.A. $z_{n-1} > z'_{n-1}$. Dann muß gelten:

$$\sum_{i=0}^{n-2} z'_i \cdot b^i - \sum_{i=0}^{n-2} z_i \cdot b^i = (z_{n-1} - z'_{n-1})\cdot b^{n-1}$$

Also gilt insbesondere

$$\sum_{i=0}^{n-2} z'_i \cdot b^i \ge b^{n-1}.$$

Dies ist ein Widerspruch zu (1).

Lösung zu Aufgabe 2

Sei die Struktur $(B, +, \cdot, \overline{\ })$ gegeben. Seien $x, y, z \in B$.
Die in (i) bis (iv) behaupteten Gleichheiten lassen sich durch das Aufstellen von Wahrheitstafeln beweisen:

(i) **Behauptung:** $(x + y) \cdot x = x$

Beweis:

x	y	$x+y$	$(\mathbf{x+y})\cdot\mathbf{x}$	$\mathbf{x}$
0	0	0	**0**	**0**
0	1	1	**0**	**0**
1	0	1	**1**	**1**
1	1	1	**1**	**1**

Behauptung: $(x \cdot y) + x = x$

Beweis:

x	y	$x\cdot y$	$(\mathbf{x\cdot y})+\mathbf{x}$	$\mathbf{x}$
0	0	0	**0**	**0**
0	1	0	**0**	**0**
1	0	0	**1**	**1**
1	1	1	**1**	**1**

(ii) **Behauptung:** $x + (y \cdot z) = (x + y) \cdot (x + z)$

Beweis:

x	y	z	$y\cdot z$	$\mathbf{x}+(\mathbf{y}\cdot\mathbf{z})$	$x+y$	$x+z$	$(\mathbf{x}+\mathbf{y})\cdot(\mathbf{x}+\mathbf{z})$
0	0	0	0	**0**	0	0	**0**
0	0	1	0	**0**	0	1	**0**
0	1	0	0	**0**	1	0	**0**
0	1	1	1	**1**	1	1	**1**
1	0	0	0	**1**	1	1	**1**
1	0	1	0	**1**	1	1	**1**
1	1	0	0	**1**	1	1	**1**
1	1	1	1	**1**	1	1	**1**

(iii) **Behauptung:** $\overline{(x + y)} = \overline{x} \cdot \overline{y}$

Beweis:

x	y	$x+y$	$\overline{(\mathbf{x+y})}$	$\overline{x}$	$\overline{y}$	$\overline{\mathbf{x}}\cdot\overline{\mathbf{y}}$
0	0	0	**1**	1	1	**1**
0	1	1	**0**	1	0	**0**
1	0	1	**0**	0	1	**0**
1	1	1	**0**	0	0	**0**

Behauptung: $\overline{(x \cdot y)} = \overline{x} + \overline{y}$

Beweis:

x	y	$x\cdot y$	$\overline{(\mathbf{x\cdot y})}$	$\overline{x}$	$\overline{y}$	$\overline{\mathbf{x}}+\overline{\mathbf{y}}$
0	0	0	**1**	1	1	**1**
0	1	0	**1**	1	0	**1**
1	0	0	**1**	0	1	**1**
1	1	1	**0**	0	0	**0**

(iv) **Behauptung:** $x = x + x = x \cdot x = \overline{\overline{x}}$

Beweis:

x	$\mathbf{x}$	$\mathbf{x}+\mathbf{x}$	$\mathbf{x}\cdot\mathbf{x}$	$\overline{x}$	$\overline{\overline{\mathbf{x}}}$
0	**0**	**0**	**0**	1	**0**
1	**1**	**1**	**1**	0	**1**

Lösung zu Aufgabe 3

Zum Beweis der Behauptung reicht es aus zu zeigen:

(i) $(z_n \dots z_2 z_1)_8 \cdot 8 = (z_n \dots z_2 z_1 0)_8$,

(ii) $\begin{aligned}(z_{n-1} \dots z_1 z_0)_8/8 &= (z_{n-1} \dots z_1)_8 \text{ Rest } z_0 \ . \\ &= (z_{n-1} \dots z_1)_8 + z_0/8\end{aligned}$

Beweis:

(i):
$$\begin{aligned}8 \cdot (z_n \dots z_2 z_1)_8 &= 8 \cdot \sum_{i=0}^{n-1} z_{i+1} \cdot 8^i \qquad \text{(nach Satz 1.1)} \\ &= \sum_{i=0}^{n-1} z_{i+1} \cdot 8^{i+1} \\ &= \sum_{j=1}^{n} z_j \cdot 8^j \\ &= 0 + \sum_{j=1}^{n} z_j \cdot 8^j \\ &= z_0 \cdot 8^0 + \sum_{j=1}^{n} z_j \cdot 8^j \qquad (\text{mit } z_0 = 0) \\ &= \sum_{j=0}^{n} z_j \cdot 8^j\end{aligned}$$

(ii):
$$\begin{aligned}(z_{n-1} \dots z_1 z_0)_8/8 &= (\sum_{i=0}^{n-1} z_i \cdot 8^i)/8 \\ &= (\sum_{i=1}^{n-1} z_i \cdot 8^i + z_0 \cdot 8^0)/8 \\ &= \sum_{i=1}^{n-1} z_i \cdot 8^{i-1} + z_0/8 \\ &= \sum_{j=0}^{n-2} z_{j+1} \cdot 8^j + z_0/8\end{aligned}$$

Lösung zu Aufgabe 4

Sei $n = 3m, m \in \mathbb{N}$.

Behauptung: $\sum_{i=0}^{n-1} z_i 2^i = \sum_{j=0}^{m-1} (z_{3j+2} z_{3j+1} z_{3j})_2 \cdot 8^j$

Beweis: Es gilt:

$$\begin{aligned}\sum_{j=0}^{m-1} (z_{3j+2} z_{3j+1} z_{3j})_2 \cdot 8^j &= \sum_{j=0}^{m-1} (z_{3j+2} 2^2 + z_{3j+1} 2^1 + z_{3j} 2^0) \cdot 2^{3j} \\ &= \sum_{j=0}^{m-1} (z_{3j+2} \cdot 2^{3j+2} + z_{3j+1} \cdot 2^{3j+1} + z_{3j} \cdot 2^{3j}) \\ &= \sum_{i=0}^{n-1} z_i 2^i, \text{ da n = 3m.}\end{aligned}$$

Lösung zu Aufgabe 5

Die Multiplikation von zwei zweistelligen Dualzahlen $a_1 a_2$ und $b_1 b_2$ nach der Schulmethode liefert folgendes Diagramm:

$(a_1$	$a_2)$	$(b_1$	$b_2)$
	a_1b_1	a_2b_1	
		a_1b_2	a_2b_2
$\mathbf{a_1b_1}$	a_1b_1	a_2b_1	a_2b_2
$\cdot$	$\oplus$	$\oplus$	
$\mathbf{a_1b_1a_2b_2}$	$\mathbf{a_1b_1a_2b_2}$	a_1b_2	

Dabei stellen die fettgedruckten Terme die Überträge dar.
Hieraus ergeben sich die folgenden Booleschen Funktionen, wobei f_1 das höchstwertige Bit des Produkts beschreibt und f_4 das niederwertigste Bit des Produkts:

$$\begin{aligned} f_1(a_1, a_2, b_1, b_2) &= a_1b_1a_2b_2 \\ f_2(a_1, a_2, b_1, b_2) &= a_1b_1 \oplus a_1b_1a_2b_2 \\ f_3(a_1, a_2, b_1, b_2) &= a_2b_1 \oplus a_1b_2 \\ f_4(a_1, a_2, b_1, b_2) &= a_2b_2 \end{aligned}$$

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

4. Mai 1998

Musterlösungen zu Blatt 2

Lösung zu Aufgabe 6

Behauptung: Die DNF einer Booleschen Funktion ist in folgendem Sinne eindeutig: Seien $I_1, I_2 \subseteq \{0, ..., 2^n - 1\}$. Dann gilt für jede Boolesche Funktion $f : B^n \to B$:

$$f(x_1, ..., x_n) = \sum_{i \in I_1} m_i(x_1, ..., x_n) = \sum_{i \in I_2} m_i(x_1, ..., x_n) \Longrightarrow I_1 = I_2$$

Beweis: Wir zeigen die Behauptung mit einem Widerspruchsbeweis:
Annahme: Es existieren zwei verschiedene Darstellungen, d.h.
$\exists I_1, I_2 \subseteq \{0, ..., 2^n - 1\}$ mit $I_1 \neq I_2$ und es gilt:

$$f = \sum_{i \in I_1} m_i = \sum_{i \in I_2} m_i.$$

Da $I_1 \neq I_2$ gilt, existiert oBdA ein Index k mit $k \in I_1$ und $k \notin I_2$. Dann gilt:

$$\begin{aligned} \sum_{i \in I_1} m_i(k) &= 1, \text{ da } k \in I_1 \\ \sum_{i \in I_2} m_i(k) &= 0, \text{ da } k \notin I_2 \end{aligned}$$

Dies ist ein Widerspruch zur Annahme.

Lösung zu Aufgabe 7

Sei $n \in \mathbb{N}, n \geq 1$.
Behauptung: Jede Boolesche Funktion $f : B^n \to B$ läßt sich darstellen als

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= \bigwedge_{\substack{(\beta_1, \ldots, \beta_n) \in B^n \\ f(\beta_1, \ldots, \beta_n) = 0}} x_1^{1-\beta_1} + x_2^{1-\beta_2} + \ldots + x_n^{1-\beta_n} \\ &= \prod_{i \in \{0,1,\ldots,2^n-1\} - I_f} M_i(x_1, x_2, \ldots, x_n). \end{aligned}$$

Beweis: Nach dem Darstellungssatz für Boolesche Funktionen (Satz 1.3 aus der Vorlesung) ist jede Boolesche Funktion $f : B^n \to B$ in disjunktiver Normalform darstellbar. Insbesondere ist also auch die Funktion

$$\overline{f} : B^n \to B, (x_1, x_2, \ldots, x_n) \mapsto \overline{f}(x_1, x_2, \ldots, x_n) := \overline{f(x_1, x_2, \ldots, x_n)}$$

darstellbar als:

$$\overline{f}(x_1, x_2, \ldots, x_n) = \sum_{i \in I_{\overline{f}}} m_i(x_1, x_2, \ldots, x_n)$$

Nun gilt aber:

$$\begin{aligned}
f(x_1, x_2, \ldots, x_n) &= \overline{\overline{f(x_1, x_2, \ldots, x_n)}} \\
&= \overline{\overline{f}(x_1, x_2, \ldots, x_n)} \\
&= \overline{\sum_{i \in I_{\overline{f}}} m_i(x_1, x_2, \ldots, x_n)} \\
&= \prod_{i \in I_{\overline{f}}} \overline{m_i(x_1, x_2, \ldots, x_n)} \qquad \text{(de Morgan)} \\
&= \prod_{i \in I_{\overline{f}}} M_i(x_1, x_2, \ldots, x_n) \qquad \text{(Lemma 1.1)}
\end{aligned}$$

Beachtet man nun, daß

$$\begin{aligned}
I_{\overline{f}} &:= \{i = (i_1, \ldots, i_n)_2 \in \{0, 1, \ldots, 2^n - 1\} | \overline{f}(i_1, \ldots, i_n) = 1\} \\
&= \{i = (i_1, \ldots, i_n)_2 \in \{0, 1, \ldots, 2^n - 1\} | f(i_1, \ldots, i_n) = 0\} \\
&= \{0, 1, \ldots, 2^n - 1\} \setminus I_f,
\end{aligned}$$

so folgt die Behauptung.

Lösung zu Aufgabe 8

Aus der Vorlesung ist bekannt, daß$\{+, \overline{\ }\}$ vollständig ist. Daher genügt es zu zeigen, daß $x + y$ und $\overline{x}$ mit Hilfe von $\downarrow$ darstellbar sind:

Behauptung:

(i) $\overline{x} = x \downarrow x$

(ii) $x + y = (x \downarrow y) \downarrow (x \downarrow y)$

Beweis: Wir beweisen (i) und (ii) durch das Aufstellen von Wahrheitstafeln:

(i):

x	$\overline{\mathbf{x}}$	$\mathbf{x \downarrow x}$
0	**1**	**1**
1	**0**	**0**

(ii):

x	y	$\mathbf{x + y}$	$x \downarrow y$	$\mathbf{(x \downarrow y) \downarrow (x \downarrow y)}$
0	0	**0**	1	**0**
0	1	**1**	0	**1**
1	0	**1**	0	**1**
1	1	**1**	0	**1**

Damit ist $\{\downarrow\}$ funktional vollständig.

Lösung zu Aufgabe 9

Sei die Boolesche Funktion $f : B^3 \to B$ gegeben durch

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

a) Darstellung von f in DNF:

$$f(x_1, x_2, x_3) = (\overline{x_1} x_2 x_3) + (x_1 x_2 \overline{x_3}) + (x_1 x_2 x_3)$$

b) Darstellung von f in KNF:

$$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x_3})(x_1 + \overline{x_2} + x_3)(\overline{x_1} + x_2 + x_3)(\overline{x_1} + x_2 + \overline{x_3})$$

c) Darstellung von f in RNF:

$$f(x_1, x_2, x_3) = (\overline{x_1} x_2 x_3) \oplus (x_1 x_2 \overline{x_3}) \oplus (x_1 x_2 x_3)$$

d) Komplementfreie Ringsummendarstellung von f:

$$\begin{aligned} f(x_1, x_2, x_3) &= (\overline{x_1} x_2 x_3) \oplus (x_1 x_2 \overline{x_3}) \oplus (x_1 x_2 x_3) \\ &= (x_1 \oplus 1) x_2 x_3 \oplus x_1 x_2 (x_3 \oplus 1) \oplus x_1 x_2 x_3 \\ &= x_1 x_2 x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_1 x_2 x_3 \\ &= x_1 x_2 x_3 \oplus x_2 x_3 \oplus x_1 x_2 \end{aligned}$$

e) Darstellung von f, die nur $\uparrow$ benutzt:
Wir zeigen zunächst die folgenden Hilfsbehauptungen:

(i) $\overline{x} y + x \overline{y} + xy = \overline{\overline{x} \cdot \overline{y}}$,

(ii) $xy = \overline{x \uparrow y}$.

Beweis der Hilfsbehauptungen:
(i):

$$\begin{aligned} \overline{x} y + x \overline{y} + xy &= \overline{x} y + x \underbrace{(\overline{y} + y)}_{=1} \\ &= \overline{x} y + x \\ &= \overline{\overline{\overline{x} y \overline{x}}} \\ &= \overline{(x + \overline{y}) \overline{x}} \\ &= \overline{\underbrace{x \overline{x}}_{=0} + (\overline{y} \cdot \overline{x})} \\ &= \overline{\overline{x} \cdot \overline{y}} \end{aligned}$$

(ii): Wir zeigen (ii) durch das Aufstellen einer Wahrheitstafel:

x	y	$\mathbf{x \cdot y}$	$x \uparrow y$	$\overline{\mathbf{x \uparrow y}}$
0	0	**0**	1	**0**
0	1	**0**	1	**0**
1	0	**0**	1	**0**
1	1	**1**	0	**1**

Mit Hilfe der Aussagen (i) und (ii) ergibt sich aus der DNF der Funktion f:

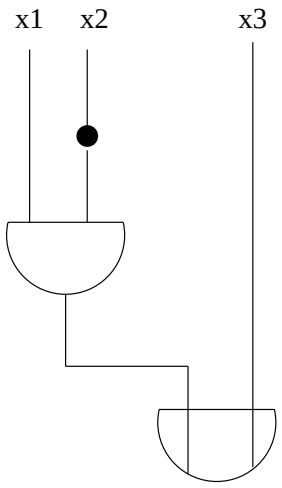
$$\begin{aligned}
f(x_1,x_2,x_3) &= (\overline{x_1}x_2x_3) + (x_1x_2\overline{x_3}) + (x_1x_2x_3) \\
&= x_2(\overline{x_1}x_3 + x_1\overline{x_3} + x_1x_3) \\
&= x_2\overline{(\overline{x_1}\cdot\overline{x_3})} \qquad \text{nach (i)} \\
&= x_2(x_1+x_3) \\
&= x_1x_2 + x_2x_3 \\
&= \overline{(x_1\uparrow x_2)} + \overline{(x_2\uparrow x_3)} \qquad \text{nach (ii)} \\
&= \overline{(x_1\uparrow x_2)(x_2\uparrow x_3)} \\
&= (x_1\uparrow x_2)\uparrow(x_2\uparrow x_3) \qquad \text{nach (ii)}
\end{aligned}$$

Lösung zu Aufgabe 10

Durch Bestimmung der DNF von f und Vereinfachung ergibt sich:

$$\begin{aligned}
f(x_1,x_2,x_3) &= \overline{x_1}\cdot\overline{x_2}\cdot x_3 + \overline{x_1}\cdot x_2\cdot x_3 + x_1\cdot\overline{x_2}\cdot\overline{x_3} + x_1\cdot\overline{x_2}\cdot x_3 + x_1\cdot x_2\cdot x_3 \\
&= \overline{x_2}\cdot x_3\cdot(\overline{x_1}+x_1) + x_2\cdot x_3\cdot(\overline{x_1}+x_1) + x_1\cdot\overline{x_2}\cdot\overline{x_3} \\
&= \overline{x_2}\cdot x_3 + x_2\cdot x_3 + x_1\cdot\overline{x_2}\cdot\overline{x_3} \\
&= x_3 + x_1\cdot\overline{x_2}\cdot\overline{x_3} \\
&= \overline{\overline{x_3}\cdot(\overline{x_1}+x_2+x_3)} \quad \text{(de Morgan)} \\
&= \overline{(\overline{x_1}\cdot\overline{x_3}) + (x_2\cdot\overline{x_3})} \\
&= \overline{\overline{x_3}\cdot(\overline{x_1}+x_2)} \\
&= x_3 + (x_1\cdot\overline{x_2})
\end{aligned}$$

Damit ergibt sich also folgendes Schaltnetz:



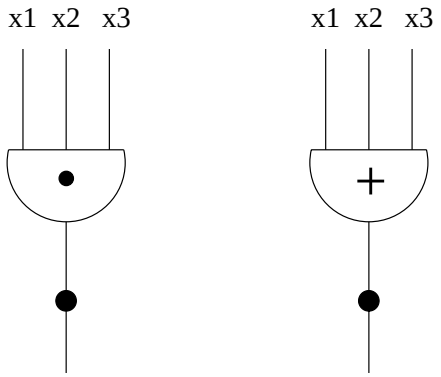
Es bleibt zu zeigen:

a) Es existiert kein Schaltnetz mit ≤ 2 Stufen, das f realisiert.

b) Es existiert kein Schaltnetz mit ≤ 2 Gattern, das f realisiert.

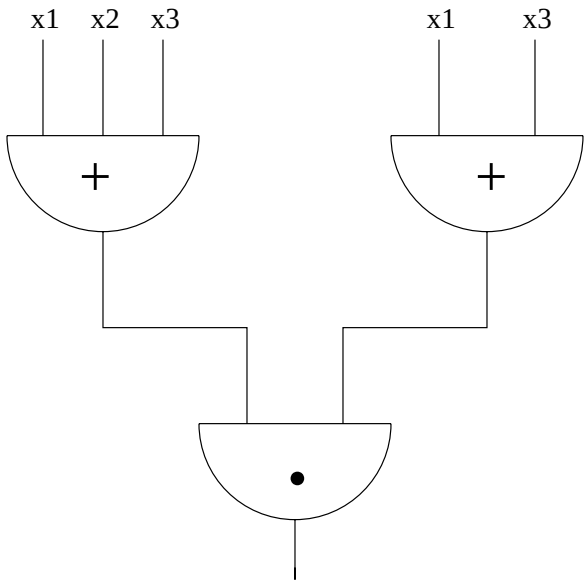
Beweis von a): Offenbar ist f mit nur einer Stufe (und somit mit nur einem Gatter) nicht realisierbar.
Mit zwei Stufen sind folgende drei Typen von Schaltnetzen möglich:

1) NOT-Gatter auf der zweiten Stufe:
   Da die Hintereinanderschaltung von zwei NOT-Gattern offenbar nicht sinnvoll ist, ergeben sich die folgenden beiden Möglichkeiten:



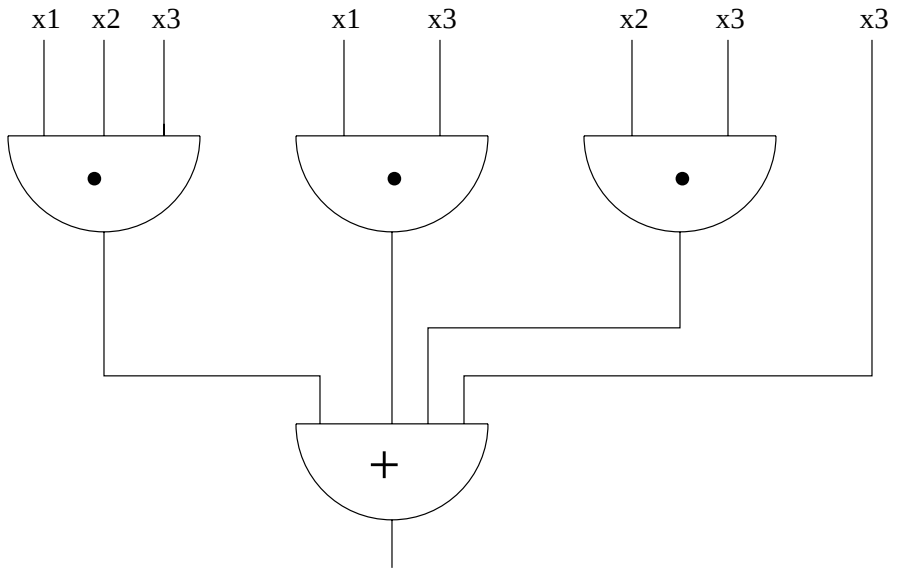
   Offensichtlich läßt sich f so jedoch nicht realisieren, da dies einer Darstellung von f mit genau einem Minterm in DNF oder mit genau einem Maxterm in KNF entspräche.

2) AND-Gatter auf der zweiten Stufe:
   AND-Gatter auf der ersten Stufe lassen sich offenbar durch direkte Eingänge in das AND-Gatter der zweiten Stufe ersetzen.
   Folgende Gatter (bzw. Eingänge) lassen sich auf der ersten Stufe jedoch nicht verwenden, da dies zu einem falschen Ergebnis bei der Verknüpfung auf der zweiten Stufe führen würde:
   $x_1, x_2, x_1 + x_2$, da $f(0,0,1) = 1$,
   $\overline{x_1}, \overline{x_2}, \overline{x_3}$, da $f(1,1,1) = 1$,
   $x_3, x_2 + x_3$, da $f(1,0,0) = 1$.
   $x_1 \cdot x_2, x_1 \cdot x_3$, da $f(0,1,1) = 1$



wobei beliebige Gatter der ersten Stufe entfernt werden können.
Offensichtlich läßt sich f so nicht realisieren, da f(1,1,0)=0 gilt.

3) OR-Gatter auf der zweiten Stufe:
OR-Gatter auf der ersten Stufe können offenbar ersetzt werden durch Eingänge in das OR-Gatter der zweiten Stufe.
Hierbei lassen sich folgende Gatter/Eingänge auf der ersten Stufe nicht verwenden:
$x_1, x_2, x_1 \cdot x_2$, da $f(1,1,0) = 0$,
$\overline{x_1}, \overline{x_2}, \overline{x_3}$, da $f(0,0,0) = 0$.
Somit ergibt sich folgendes Schaltnetz,



wobei ebenfalls beliebige Gatter der ersten Stufe bzw. Eingänge des OR-Gatters entfernt werden können.
Offensichtlich realisiert auch keines dieser Netze f, da f(1,0,0)=1 gilt.

Beweis von b): Diese Behauptung folgt direkt aus a): Da das Schaltnetz aus mindestens drei Stufen aufgebaut werden muß, kann es auch nicht weniger als drei Gatter enthalten.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

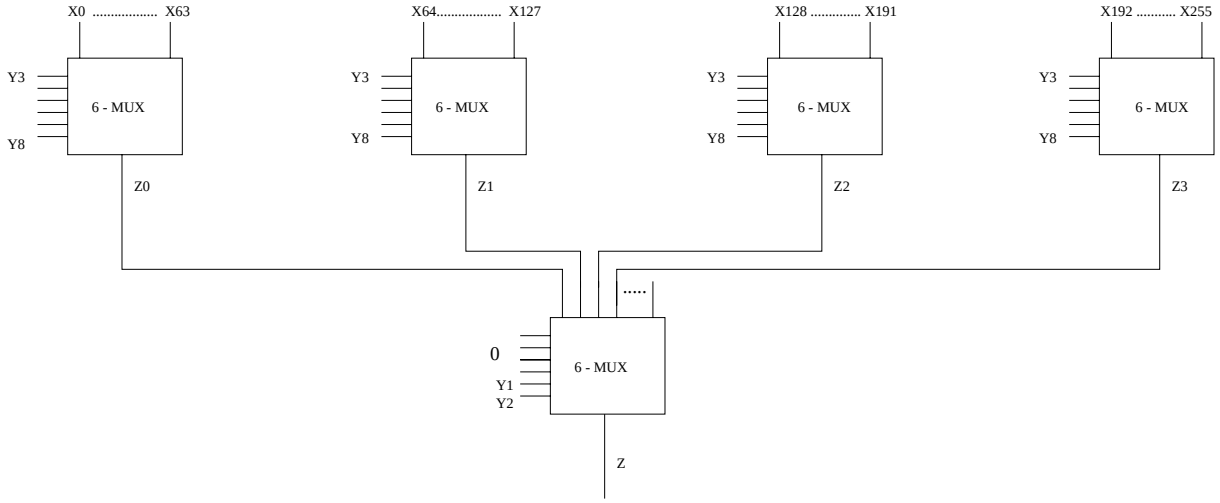
11. Mai 1998

Musterlösungen zu Blatt 3

Lösung zu Aufgabe 11

a) Konstruktion:

Aus gegebenen Bauteilen mit der Funktion eines 6-MUX kann man folgendermaßen einen 8-MUX konstruieren:



b) Korrektheit der Konstruktion:

Zu zeigen: $Z(X_0, X_1, \ldots, X_{2^8-1}, Y_1, \ldots, Y_8) = \bigvee_{(\alpha_1,\ldots,\alpha_8)\in B^8} X_{(\alpha_1,\ldots,\alpha_8)_2} Y_1^{\alpha_1} Y_2^{\alpha_2} \cdots Y_8^{\alpha_8}$

Beweis: Es gilt:

$$
\begin{aligned}
Z &= \bigvee_{(\alpha_1,\alpha_2)\in B^2} Y_1' Y_2' Y_3' Y_4' Y_1^{\alpha_1} Y_2^{\alpha_2} Z_{(0000\alpha_1\alpha_2)_2} \\
Z &= \bigvee_{(\alpha_1,\alpha_2)\in B^2} Y_1^{\alpha_1} Y_2^{\alpha_2} Z_{(\alpha_1\alpha_2)_2} \\
&= \bigvee_{(\alpha_1,\alpha_2)\in B^2} Y_1^{\alpha_1} Y_2^{\alpha_2} X_{(\alpha_1\alpha_2)_2\cdot 2^6+(Y_3\ldots Y_8)_2} \\
&= \bigvee_{(\alpha_1,\alpha_2)\in B^2} Y_1^{\alpha_1} Y_2^{\alpha_2} \left(\bigvee_{(\alpha_3,\ldots,\alpha_8)\in B^6} \left(X_{(\alpha_1\alpha_2)_2\cdot 2^6+(\alpha_3\ldots\alpha_8)_2} Y_3^{\alpha_3}\cdots Y_8^{\alpha_8} \right) \right) \\
&= \bigvee_{(\alpha_1,\alpha_2)\in B^2} Y_1^{\alpha_1} Y_2^{\alpha_2} \left(\bigvee_{(\alpha_3,\ldots,\alpha_8)\in B^6} \left(X_{(\alpha_1\ldots\alpha_8)_2} Y_3^{\alpha_3}\cdots Y_8^{\alpha_8} \right) \right) \\
&= \bigvee_{(\alpha_1,\ldots,\alpha_8)\in B^8} X_{(\alpha_1\ldots\alpha_8)_2} Y_1^{\alpha_1} Y_2^{\alpha_2} Y_3^{\alpha_3}\cdots Y_8^{\alpha_8}
\end{aligned}
$$

c) Komplexität (Anzahl der Gatter) des 6-MUX:
Gemäß der Vorlesung kann man einen 6-MUX mit $3 \cdot (2^6 - 1) = 189$ AND- und OR-Gattern sowie 6 Invertern realisieren. Die Konstruktion aus a) benötigt fünf 6-MUX, jedoch insgesamt nur 8 Inverter, also insgesamt $5 \cdot 189 + 8 = 953$ Gatter.

Lösung zu Aufgabe 12

a) Rekursive Konstruktion: Ein 1-MUX wird konstruiert, wie in der Vorlesung beschrieben.
Nach 1.2.2. in der Vorlesung kann ein $2d$-MUX aus d-MUX-Bauteilen konstruiert werden. Führt man diese Konstruktion m-mal rekursiv aus, so kann man damit einen 2^{4m}-MUX aus 2^{3m}-MUX-Bauteilen konstruieren.

b) Korrektheit *mittels vollständiger Induktion:*

$m = 0$: trivialerweise erfüllt.

$m \to m + 1$: Die Konstruktion d-MUX $\to$ $2d$-MUX wurde in der Vorlesung verifiziert. D.h. nach jedem Iterationsschritt erhält man einen MUX mit doppelt so vielen Dateneingängen. Folglich hat man nach den m Iterationen aus a) einen korrekt arbeitenden $2^{m-1} \cdot 2^{3m} = 2^{4m}$-MUX konstruiert.

c) Komplexität: Ein nach der Konstruktion aus 1.2.2 der Vorlesung aufgebauter d-MUX besteht aus $d + 3 \cdot (2^d - 1)$ Gattern.
Da in a) genau diese Konstruktion angewendet wurde, besteht der konstruierte 2^{4m}- MUX also aus $2^{4m} + 3 \cdot (2^{2^{4m}} - 1)$ Gattern.

Lösung zu Aufgabe 13

Um für $d \geq 3$ eine d-stellige Boolesche Funktion f mit Hilfe eines $(d-1)$-MUX zu realisieren, konstruiere eine Funktion $\tilde{f}(x_1, x_2, \cdots, x_{d-1})$ aus der Funktion $f(x_1, x_2, \cdots, x_d)$ wie folgt:

$$\tilde{f}(x_1, \ldots, x_{d-1}) = \begin{cases} 0 & \text{falls } f(x_1, \ldots, x_{d-1}, 0) = f(x_1, \ldots, x_{d-1}, 1) = 0 \\ 1 & \text{falls } f(x_1, \ldots, x_{d-1}, 0) = f(x_1, \ldots, x_{d-1}, 1) = 1 \\ x_d & \text{falls } f(x_1, \ldots, x_{d-1}, x_d) = x_d \\ \overline{x_d} & \text{falls } f(x_1, \ldots, x_{d-1}, x_d) = \overline{x_d} \end{cases}$$

Wenn man die Werte von $\tilde{f}(0, \ldots, 0), \ldots, \tilde{f}(1, \ldots, 1)$ als Eingangsdaten und $x_1, x_2, \cdots, x_{d-1}$ als Steuersignale für einen $(d-1)$-MUX benutzt, dann berechnet der $(d-1)$-MUX genau die Funktion f.

Berechnung der Anzahl der benötigten Gatter:
Ein $(d-1)$-MUX kann mit $d - 1 + 3 \cdot (2^{d-1} - 1)$ Gattern realisiert werden (wie in der Vorlesung bewiesen). Zusätzlich wird noch ein Inverter benötigt, um $\overline{x_d}$ zu berechnen.
Insgesamt ist die Anzahl der benötigten Gatter also höchstens:

$$d + 3 \cdot (2^{d-1} - 1).$$

Lösung zu Aufgabe 14

Wir zeigen die Aussage für $c = 2$:

Behauptung: Für $d \geq 9$ existiert eine d-stellige Boolesche Funktion f, so daß jedes Schaltnetz für f mindestens

$$2 \cdot \frac{2^{d-3}}{d}$$

Knoten hat.

Beweis: In der Vorlesung wurde der Beweis der Abschätzung für $c = 1$ über die Länge einer Kodierung $\mathrm{Kod}(S)$ des Schaltnetzes S geführt.
Idee: Verkürzung der Kodierung, d.h. Definition einer neuen Kodierung $\mathrm{Kod}'(S)$ mit $|\mathrm{Kod}'(S)| < |\mathrm{Kod}(S)|$
Seien die Knoten als $v_0, ..., v_d, ..., v_{m-1}$ numeriert. Kodiere wie folgt:

1. Falls v ein Input (d. h. ein Gatter mit $\mathrm{indeg}(v) = 0$) ist, dann sei $\mathrm{Kod}'(v) = (11x)_2$, wobei mit x der Index des Eingangs kodiert wird (mit $|x| = \lceil \log_2 d \rceil$).
   $\Rightarrow |\mathrm{Kod}'(v)| = 2 + \lceil \log_2 d \rceil$

2. Falls v ein Gatter mit $\mathrm{indeg}(v) = 1$, dann sei $\mathrm{Kod}'(v) = (10xe)_2$, wobei mit x die gewünschte einstellige Boolesche Funktion und mit e der Eingangsknoten kodiert wird (mit $|e| = \lceil \log_2 m \rceil$).
   $\Rightarrow |\mathrm{Kod}'(v)| = 4 + \lceil \log_2 m \rceil$

3. Falls v ein Gatter mit $\mathrm{indeg}(v) = 2$, dann sei $\mathrm{Kod}'(v) = (0de_1e_2)_2$, wobei d die gewünschte 2-stellige Boolesche Funktion und e_1 und e_2 die Eingänge kodieren (mit $|d| = 4, |e_1| = |e_2| = \lceil \log_2 m \rceil$).
   $\Rightarrow |\mathrm{Kod}'(v)| = 5 + 2 \cdot \lceil \log_2 m \rceil$

Daraus folgt, daß

$$|\mathrm{Kod}'(S)| \leq m \cdot (5 + 2 \cdot \lceil \log_2 m \rceil)$$

$\Rightarrow 2^{m \cdot (5+2 \cdot \lceil \log_2 m \rceil)+1} \geq 2^{2^d}$ (analog zur Vorlesung)
$\Rightarrow m \cdot (5 + 2 \cdot \lceil \log_2 m \rceil) + 1 \geq 2^d$

Um $m \geq 2 \cdot \frac{2^{d-3}}{d}$ zu zeigen, reicht es aus,

$$\frac{2^{d-2}}{d} \cdot (5 + 2\lceil \log_2(\frac{2^{d-2}}{d}) \rceil) + 1 < 2^d$$

zu zeigen. Es gilt:

$$\begin{aligned}
\tfrac{2^{d-2}}{d} \cdot (5 + 2\lceil \log_2(\tfrac{2^{d-2}}{d}) \rceil) + 1 &\leq \tfrac{2^{d-2}}{d} \cdot (7 + 2((d-2) - \log_2 d)) + 1 \\
&\leq \tfrac{1}{d}(7 \cdot 2^{d-2} + 2^{d-1}(d - 2 - \log_2 d)) + 1 \\
&\leq \tfrac{7}{d} \cdot 2^{d-2} + \tfrac{2}{d} 2^{d-2} \cdot d + 1 \\
&\leq \tfrac{7}{9} 2^{d-2} + 2^{d-1} + 1 \\
&\leq 2^{d-2} + 2^{d-1} + 1 \\
&< 2^d
\end{aligned}$$

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

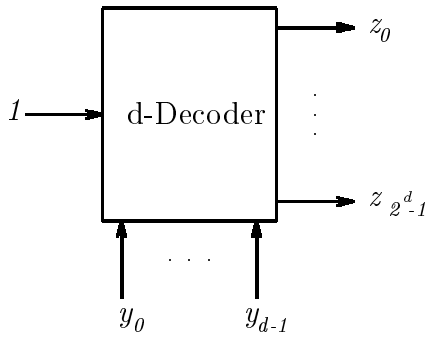
Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

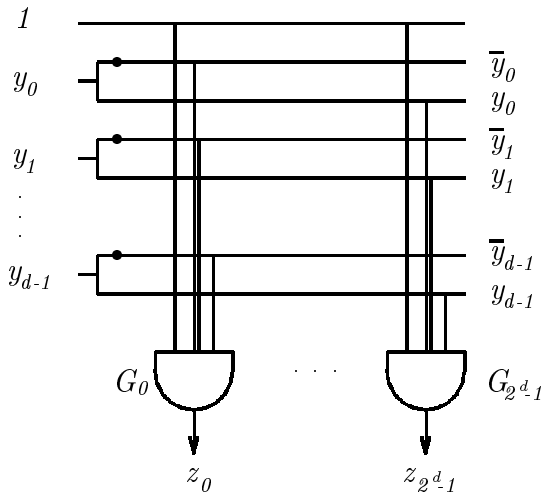
18. Mai 1998

Musterlösungen zu Blatt 4

Lösung zu Aufgabe 15



Es werden 2^d UND-Gatter mit einem $(d+1)$-Fan-In und d Inverter zur Invertierung der Dateneingänge $y_0, \ldots, y_{d-1}$ benötigt. Jedes der UND-Gatter G_j ist mit dem Dateninput 1 und für $i = 1, \cdots, d$ entweder mit y_i oder $\overline{y_i}$ verbunden. Für das Gatter G_j mit $j = (j_{d-1}, \cdots, j_0)_2$ mit $(0 \leq j \leq 2^d - 1)$ wähle folgende Belegung:
Eingang i $(0 \leq i \leq d-1)$: Ist mit y_i verbunden, falls $j_i = 1$, anderenfalls mit $\overline{y_i}$.
Eingang d: Hier liegt das Steuersignal 1 an.



Lösung zu Aufgabe 16

a) *(i) Anzahl der benötigten Gatter:*

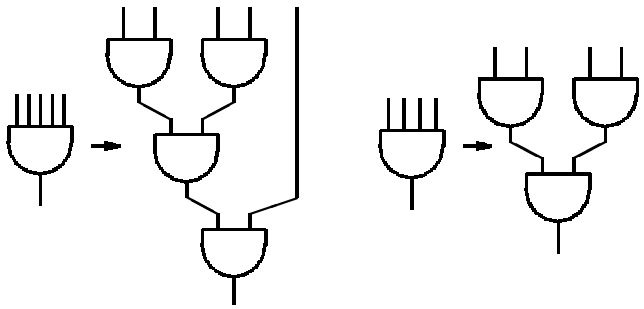
Baustein	Anzahl	verwendete Gatter	Σ
$\tilde{A}_4$	2	siehe I.2.3.: $5 \cdot 4$ (Ripple-Carry-Adder mit zusätzlichem Übertrag als Eingang, Existenz 1-stufiger $\oplus$-Gatter vorausgesetzt)	40
A_4	1	20 [für den $\tilde{A}_4$-Baustein] + 8 [in 1. Stufe Carry-Select] + 1 + 2 + 3 + 4 [2-stufige (!) UND-Gatter] + 4 [2-stufige ODER Gatter für Carry]	42
$R_{4..8}$	1	1 [Inverter] + $5 \cdot$ (2 [UND] + 1 [ODER])	16

Insgesamt werden also 98 Gatter benötigt.

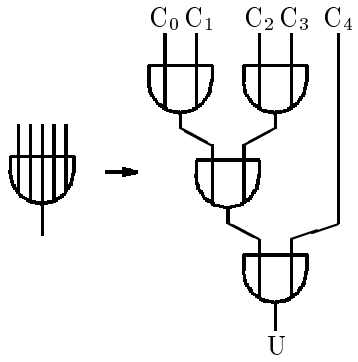
ii) Zeitkomplexität des Addiernetzes:

Um die Zeitkomplexität des Netzes zu ermitteln, ist es ausreichend, zu bestimmen, wann das Ergebnis des Ausgangs R_8 anliegt. (Offensichtlich haben alle anderen Ausgänge eine geringere Komplexität.) Nach I.2.3. liefert ein Ripple-Carry-Adder sein Ergebnis in $2 \cdot n - 1$ ZE. Da hier auch an der ersten Stelle schon ein Übertrag berücksichtigt werden muß, liefert $\tilde{A}_4$ seine Ergebnisse nach $2 \cdot n + 1$ ZE. Das bedeutet, daß die Ergebnisse $v_8, \cdots, v_4$ bzw. $u_8, \cdots, u_4$ nach 9 ZE verfügbar sind. Die Bestimmung von U mittels Carry-Select geschieht in 6 ZE. Benennt man nämlich die fünf Eingänge in das (große) ODER-Gatter (siehe Vorlesung $\rightarrow$ *2-stelliges Carry-Select-Addierwerk*) beim Carry-Select mit $C_0, \cdots, C_4$, so ergeben sich folgende Zeiten für die Bereitstellung eines gültigen Wertes:

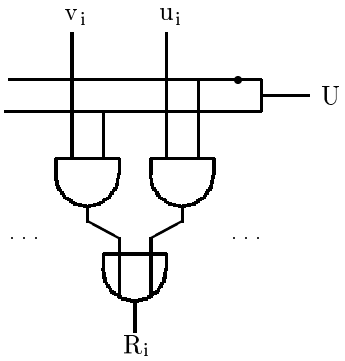
Eingang	C_0	C_1	C_2	C_3	C_4
Komplexität [ZE]	1	2	3	3	4



Geschickte Bildung von $C_0 + C_1 + C_2 + C_3 + C_4$ (siehe Skizze) liefert U nach 6 ZE, also noch vor dem Zeitpunkt, zu dem die Ergebnisse der $\tilde{A}_4$-Bausteine vorliegen, so daß $\overline{U}$ nach 7 ZE vorliegt.



Folglich müssen die Ergebnisse nur noch den Baustein $R_{4..8}$ durchlaufen. Da $\overline{U}$ bereits bestimmt ist, werden noch zwei weitere ZE benötigt (siehe Skizze).



Folglich steht das Ergebnis nach 11 ZE.

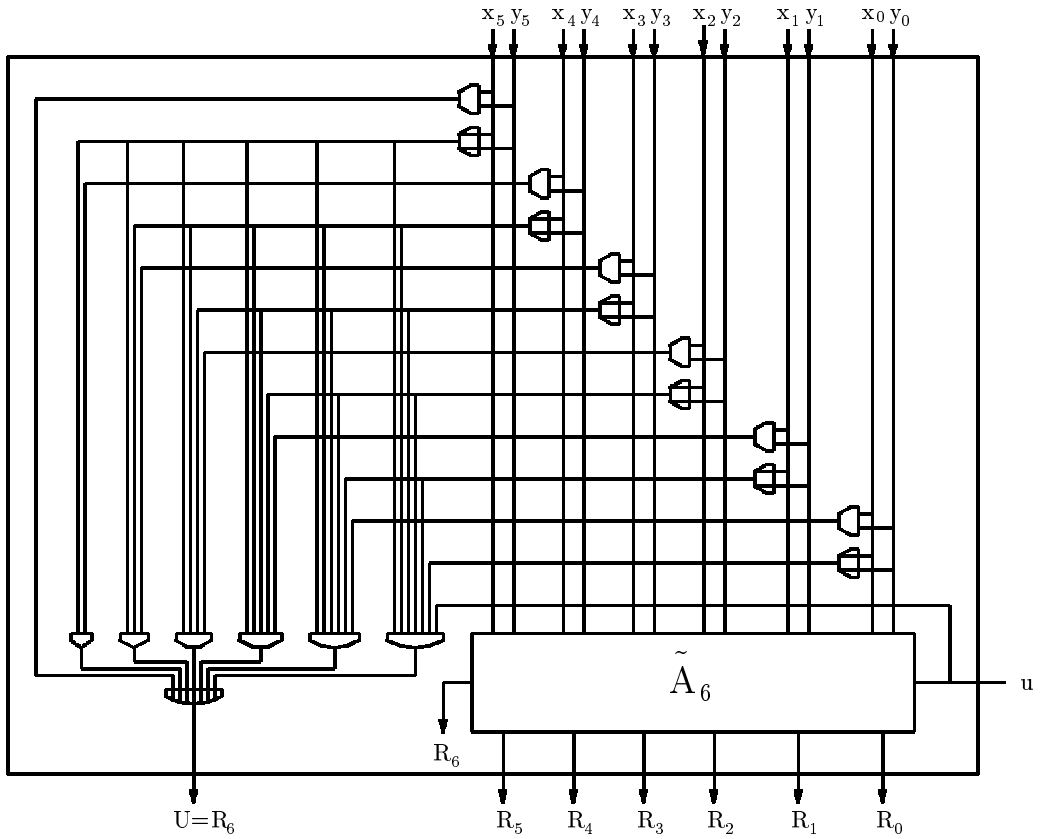
b) 12-stelliges Carry-Select-Addiernetz:

Die Idee kann direkt aus der Vorlesung übernommen werden. Der Aufbau eines Ripple-Carry-Adder $\tilde{A}_n$-Gatters mit Übertrag als zusätzlicher Eingabe (insbesondere hier für n = 6) ist bekannt. Bleibt also noch die Konstruktion eines Carry-Look-Ahead-Bausteins A_6.

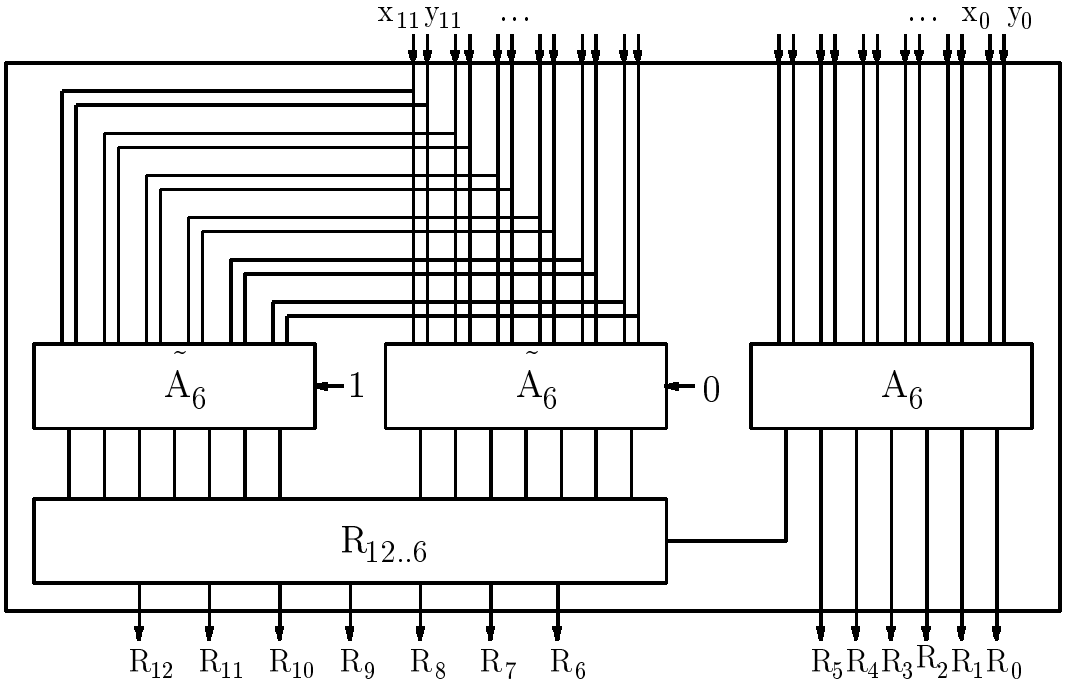
In diesem Fall berechnet sich der Übertrag U zu:

$$\begin{aligned} U \;=\; & x_5 \cdot y_5 + (x_5 + y_5) \cdot x_4 \cdot y_4 + (x_5 + y_5) \cdot (x_4 + y_4) \cdot x_3 \cdot y_3 \\ & + (x_5 + y_5) \cdot (x_4 + y_4) \cdot (x_3 + y_3) \cdot x_2 \cdot y_2 \\ & + (x_5 + y_5) \cdot (x_4 + y_4) \cdot (x_3 + y_3) \cdot (x_2 + y_2) \cdot x_1 \cdot y_1 \\ & + (x_5 + y_5) \cdot (x_4 + y_4) \cdot (x_3 + y_3) \cdot (x_2 + y_2) \cdot (x_1 + y_1) \cdot x_0 \cdot y_0 \\ & + (x_5 + y_5) \cdot (x_4 + y_4) \cdot (x_3 + y_3) \cdot (x_2 + y_2) \cdot (x_1 + y_1) \cdot (x_0 + y_0) \cdot u \end{aligned}$$

Das Schaltbild für A_n sieht also wie folgt aus:



Der Aufbau des Bausteins $R_{12..6}$ ist klar. Folglich:



Benennt man die sieben Eingänge in das (große) ODER-Gatter beim Carry-Select mit $C_0, \cdots, C_6$, so ergibt sich folgende Komplexität:

Eingang	C_0	C_1	C_2	C_3	C_4	C_5	C_6
Komplexität [ZE]	1	2	3	3	4	4	4

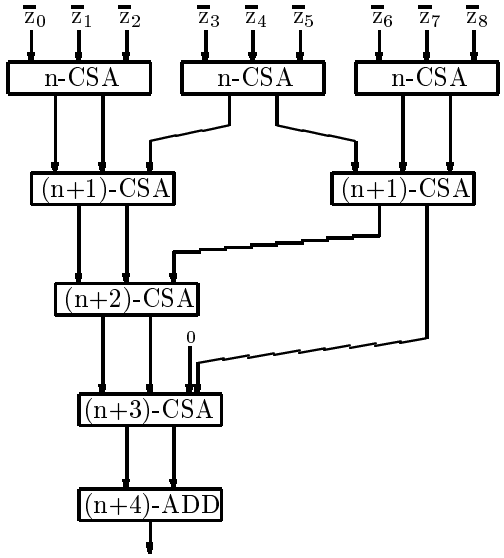
Geeigneter Einsatz von ODER-Gattern liefert $C_0 + \cdots + C_6$ in drei ZE, so daß $R_{12..6}$ nach insgesamt 8 ZE sowohl mit U als auch $\overline{U}$ beliefert wurde. Die Ergebnisse der $\tilde{A}_6$-Bausteine

sind nach $2 \cdot 6 + 1 = 13$ ZE gültig, so daß das Endergebnis nach Durchlauf durch $R_{12..6}$ nach 15 ZE ausgegeben wird.

Dieses Ergebnis läßt sich noch dadurch verbessern, daß man statt der Ripple-Carry-Addierer $\tilde{A}_6$ 6-stellige Carry-Select-Addierer verwendet. Mit analoger Rechnung wie oben ergibt sich, daß diese die Ergebnisse nach 9 ZE liefern, das Endergebnis liegt in diesem Fall also nach 11 ZE vor.

Lösung zu Aufgabe 17

Hier bietet sich die Verwendung von CSA-Bausteinen an.



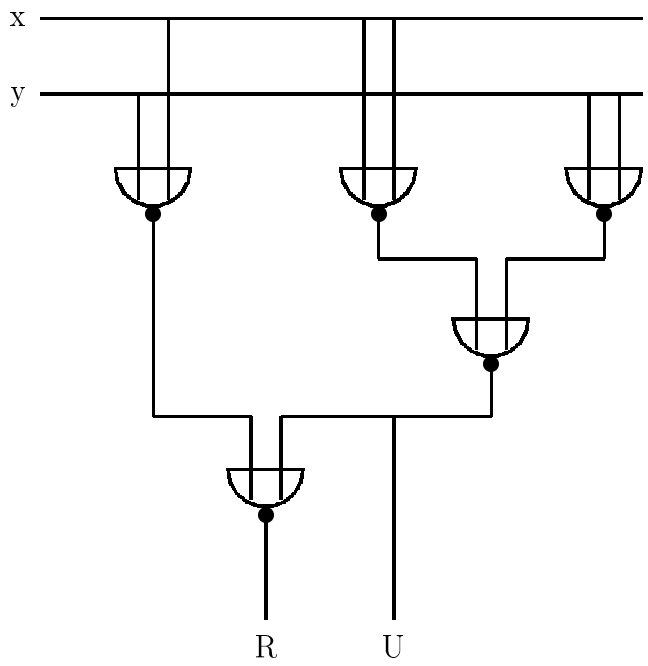
Diese reduzieren das Problem der Addition von neun n-stelligen Dualzahlen zunächst auf das Problem sechs (n+1)-stellige Dualzahlen zu addieren. Im nächsten Schritt kann dieses Problem auf die Addition von vier (n+2)-stelligen Zahlen heruntergebrochen werden werden, die jetzt in zwei Schritten addiert werden. Schließlich werden die resultierenden zwei (n+4)-Zahlen addiert. Es existiert keine günstigere Berechnung mit Hilfe von CSAs, da die maximal dadurch erzielbare Reduktion $3 \rightarrow 2$ Additionen in jeder Ebene erreicht wird.
Die Gesamtkomplexität berechnet sich zu:
$3 + 3 + 3 + 3$ [CSA] + $\lceil log_2(n+4) \rceil$ [ADD] ZE.

Lösung zu Aufgabe 18

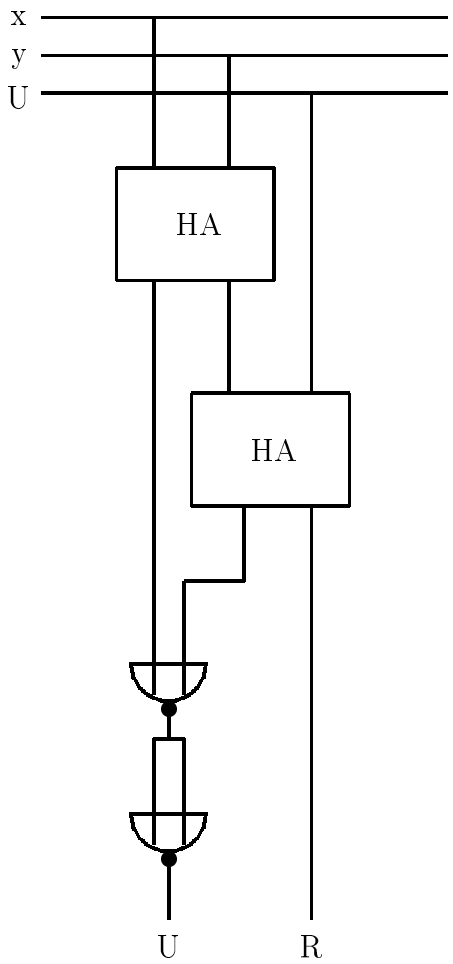
(a) Um einen Halbaddierer aus NOR-Gattern zu konstruieren, reicht es aus, ein UND-Gatter und ein XOR-Gatter mit NOR-Gattern wie folgt zu simulieren. Es gilt:

$$
\begin{aligned}
\overline{x} &= x \downarrow x \\
x + y &= (x \downarrow y) \downarrow (x \downarrow y) \\
x \cdot y &= (x \downarrow x) \downarrow (y \downarrow y) \\
x \oplus y &= (\overline{x} + \overline{y})(x + y) && \text{* Maxterm von XOR *} \\
&= ((\overline{x} + \overline{y}) \downarrow (\overline{x} + \overline{y})) \downarrow ((x + y) \downarrow (x + y)) && \text{* siehe oben *} \\
&= ((\overline{x} + \overline{y}) \downarrow (\overline{x} + \overline{y})) \downarrow \overline{(x + y)} && \text{* siehe oben *} \\
&= ((\overline{x} + \overline{y}) \downarrow (\overline{x} + \overline{y})) \downarrow (x \downarrow y) && \text{* } \overline{(x + y)} = x \downarrow y \text{ *} \\
&= \overline{(\overline{x} + \overline{y})} \downarrow (x \downarrow y) && \text{* siehe oben *} \\
&= (\overline{x} \downarrow \overline{y}) \downarrow (x \downarrow y) && \text{* } \overline{(x + y)} = x \downarrow y \text{ *} \\
&= ((x \downarrow x) \downarrow (y \downarrow y)) \downarrow (x \downarrow y) && \text{* siehe oben *}
\end{aligned}
$$

Damit ergibt sich folgendes Schaltbild:



(b) Für einen ausschließlich aus NOR-Gattern bestehenden Volladdierer ergibt sich unter Verwendung der in a) konstruierten Halbaddierer folgendes Schaltbild:



(c) Es ergeben sich folgende Schaltzeiten:

Resultat des Halbaddierers:	60 psec
Übertrag des Halbaddierers:	40 psec
Resultat des Volladdierers:	120 psec
Übertrag des Volladdierers:	140 psec

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

25. Mai 1998

Musterlösungen zu Blatt 5

Lösung zu Aufgabe 19

a): Es gilt:

$$\begin{aligned} f &= \underbrace{\overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3 \cdot \overline{x}_4}_{f_1} + \underbrace{\overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3 \cdot x_4}_{f_2} + \underbrace{\overline{x}_1 \cdot \overline{x}_2 \cdot x_3 \cdot x_4}_{f_3} + \\ &\quad \underbrace{x_1 \cdot \overline{x}_2 \cdot \overline{x}_3 \cdot \overline{x}_4}_{f_4} + \underbrace{x_1 \cdot \overline{x}_2 \cdot \overline{x}_3 \cdot x_4}_{f_5} + \underbrace{x_1 \cdot \overline{x}_2 \cdot x_3 \cdot x_4}_{f_6} + \\ &\quad \underbrace{x_1 \cdot x_2 \cdot \overline{x}_3 \cdot \overline{x}_4}_{f_7} + \underbrace{x_1 \cdot x_2 \cdot \overline{x}_3 \cdot x_4}_{f_8} + \underbrace{x_1 \cdot x_2 \cdot x_3 \cdot x_4}_{f_9} \\ &= \underbrace{\overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}_3}_{f_1+f_2=g_1} + \underbrace{\overline{x}_1 \cdot \overline{x}_2 \cdot x_4}_{f_2+f_3=g_2} + \underbrace{x_1 \cdot \overline{x}_2 \cdot \overline{x}_3}_{f_4+f_5=g_3} + \\ &\quad \underbrace{x_1 \cdot \overline{x}_2 \cdot x_4}_{f_5+f_6=g_4} + \underbrace{x_1 \cdot x_2 \cdot \overline{x}_3}_{f_7+f_8=g_5} + \underbrace{x_1 \cdot x_2 \cdot x_4}_{f_8+f_9=g_6} \\ &= \underbrace{\overline{x}_2 \cdot \overline{x}_3}_{g_1+g_3=h_1} + \underbrace{\overline{x}_2 \cdot x_4}_{g_2+g_4=h_2} + \underbrace{x_1 \cdot \overline{x}_3}_{g_3+g_5=h_3} + \underbrace{x_1 \cdot x_4}_{g_4+g_6=h_4} \end{aligned}$$

Ganz offensichtlich kann man f mit der Karnaugh-Methode nicht mehr weiter optimieren.
b) Dennoch ist diese Darstellung noch nicht optimal im Sinne der Anzahl der verwendeten zweistelligen Operatoren, denn es gilt:

$$\begin{aligned} f &= \underbrace{\overline{x}_3 \cdot (\overline{x}_2 + x_1)}_{h_1+h_3=i_1} + \underbrace{x_4 \cdot (\overline{x}_2 + x_1)}_{h_2+h_4=i_2} \\ &= \underbrace{(\overline{x}_3 + x_4) \cdot (\overline{x}_2 + x_1)}_{i_1+i_2} \end{aligned}$$

Diese Darstellung benötigt nur 3 statt 7 zweistellige Operatoren.

Lösung zu Aufgabe 20

a): Zunächst werden alle Implikanten von f bestimmt. Zur Veranschaulichung empfiehlt es sich, die Funktionswerte in ein Karnaugh-Diagramm einzutragen:

		0	0	1	1	x_3
x_1	x_2	0	1	1	0	x_4
0	0	0	0	1	1	
0	1	0	1	1	0	
1	1	1	1	0	0	
1	0	1	0	0	1	

Aus dieser Übersicht kann nun leicht das Quine-McCluskey-Verfahren formal durchgeführt werden:

$$
\begin{array}{lcl@{\qquad}lcl}
f_0 & = & \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 \cdot x_4 & f_0 + f_1 & = & g_0 = \overline{x}_1 \cdot x_3 \cdot x_4 \\
f_1 & = & \overline{x}_1 \cdot x_2 \cdot x_3 \cdot x_4 & f_1 + f_2 & = & g_1 = \overline{x}_1 \cdot x_2 \cdot x_4 \\
f_2 & = & \overline{x}_1 \cdot x_2 \cdot \overline{x}_3 \cdot x_4 & f_2 + f_3 & = & g_2 = x_2 \cdot \overline{x}_3 \cdot x_4 \\
f_3 & = & x_1 \cdot x_2 \cdot \overline{x}_3 \cdot x_4 & f_3 + f_4 & = & g_3 = x_1 \cdot x_2 \cdot \overline{x}_3 \\
f_4 & = & x_1 \cdot x_2 \cdot \overline{x}_3 \cdot \overline{x}_4 & f_4 + f_5 & = & g_4 = x_1 \cdot \overline{x}_3 \cdot \overline{x}_4 \\
f_5 & = & x_1 \cdot \overline{x}_2 \cdot \overline{x}_3 \cdot \overline{x}_4 & f_5 + f_6 & = & g_5 = x_1 \cdot \overline{x}_2 \cdot \overline{x}_4 \\
f_6 & = & x_1 \cdot \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 & f_6 + f_7 & = & g_6 = \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 \\
f_7 & = & \overline{x}_1 \cdot \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 & f_7 + f_0 & = & g_7 = \overline{x}_1 \cdot \overline{x}_2 \cdot x_3
\end{array}
$$

Da das Verfahren keine weiteren Vereinfachungen zuläßt, sind dies alle Implikanten.
b): Die Primimplikanten lassen sich leicht ablesen:

$$
\begin{array}{ll}
\overline{x}_1 \cdot x_3 \cdot x_4 & \overline{x}_1 \cdot x_2 \cdot x_4 \\
x_2 \cdot \overline{x}_3 \cdot x_4 & x_1 \cdot x_2 \cdot \overline{x}_3 \\
x_1 \cdot \overline{x}_3 \cdot \overline{x}_4 & x_1 \cdot \overline{x}_2 \cdot \overline{x}_4 \\
\overline{x}_2 \cdot x_3 \cdot \overline{x}_4 & \overline{x}_1 \cdot \overline{x}_2 \cdot x_3
\end{array}
$$

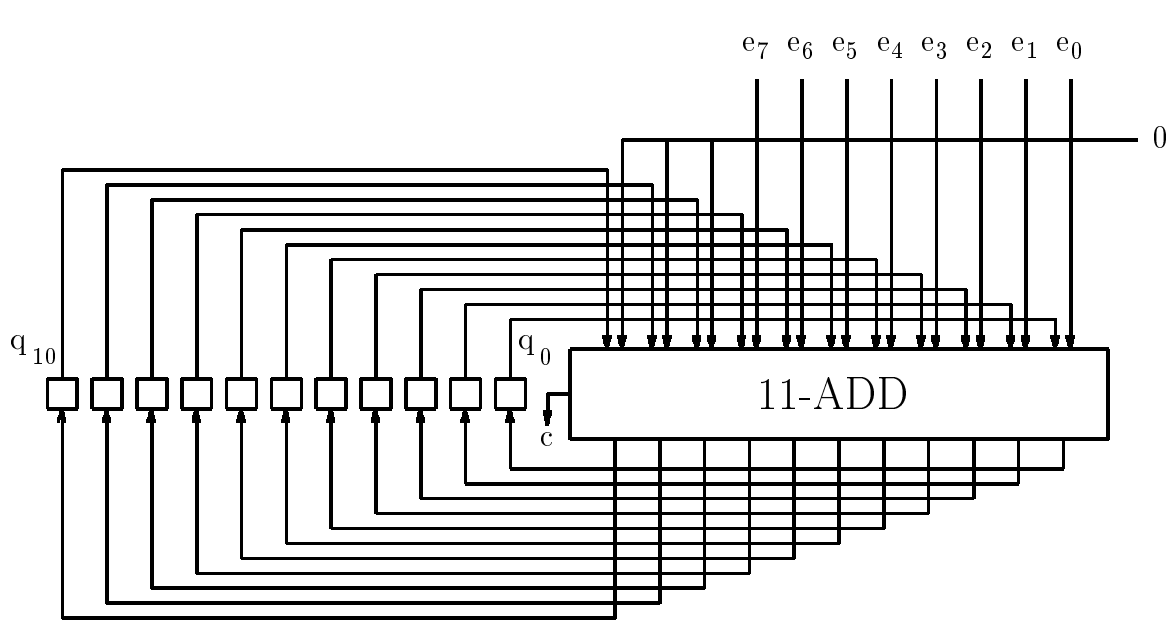
Wie man auch dem Karnaugh-Diagramm entnehmen kann, sind dies alle Primimplikanten.
c): Wähle als Darstellung etwa:

$$
\begin{array}{lcl}
f_1 & = & \overline{x}_1 \cdot x_3 \cdot x_4 + x_2 \cdot \overline{x}_3 \cdot x_4 + x_1 \cdot \overline{x}_3 \cdot \overline{x}_4 + \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 \\
f_2 & = & \overline{x}_1 \cdot x_2 \cdot x_4 + x_1 \cdot x_2 \cdot \overline{x}_3 + x_1 \cdot \overline{x}_2 \cdot \overline{x}_4 + \overline{x}_1 \cdot \overline{x}_2 \cdot x_3
\end{array}
$$

Da jeder Primimplikant genau von zwei Mintermen impliziert wird und es acht Minterme gibt, sind diese Darstellungen optimal. Eine analoge Begründung wäre: Da die zugehörigen Primblöcke im Karnaugh-Diagramm disjunkt sind (nur horizontale bzw. nur vertikale Blöcke), sind diese Darstellungen optimal.

Lösung zu Aufgabe 21

a): Die Idee ist es, die einzelnen Summanden sukzessive zu addieren. Dazu geht man davon aus, das die Delays zu Beginn mit Nullen gefüllt sind. Dieser Inhalt (die Zahl 0) ist dann der erste Summand, zu dem die erste der 8-stelligen Dualzahlen addiert wird. Das Ergebnis wird in den Delays gespeichert und dient erneut als Operand für die nächste Addition. Das Schaltnetz sieht folgendermaßen aus:



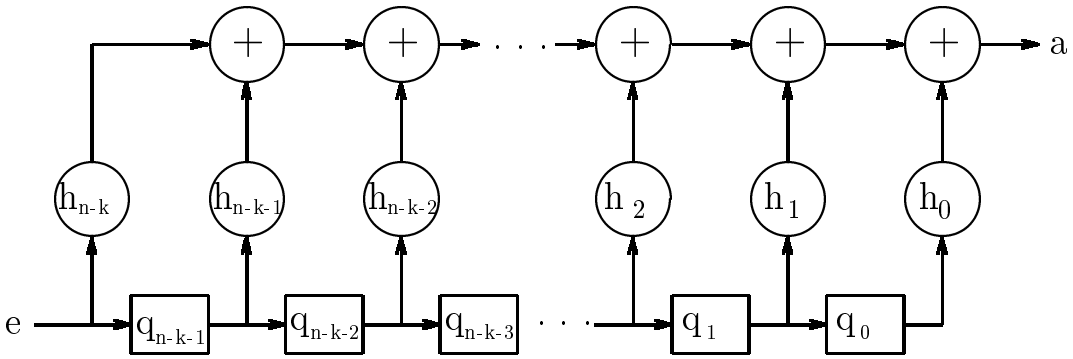
b): Der zugehörige Mealy-Automat $A = (Q, \Sigma, \Delta, q_s, F, \delta)$ ist definiert durch:

Tupelelement	**Idee**
$Q = B^{11}$	Wir benötigen 11 Register, um das Ergebnis (und die Zwischenwerte) zu speichern.
$\Sigma = B^8$	Die Eingabewerte sind 8-stellige Dualzahlen.
$\Delta = B^{11}$	Nach jeder Addition soll der komplette Inhalt der Register (Delays) ausgegeben werden.
$q_s = (0, \ldots, 0)$	Zu Beginn sind alle Delays gelöscht, enthalten also den Wert 0 (hierauf werden die Summanden sukkzessive addiert).
$F = \emptyset$	Wir benötigen keinen ausgezeichneten Zustand.
δ	$$\delta : Q \times \Sigma \to Q \times \Delta, (q, e) \mapsto (q \oplus e, q \oplus e).$$ Dabei ist mit $\oplus$ jeweils die Addition der binären Zahlen in $\mathbb{F}_{2^{12}}$ (= Abschneiden des Übertrags) gemeint, also: $$\delta : (q = (q_{10}, \ldots, q_0), e = (e_7, \ldots, e_0)) = (\tilde{q} = (\tilde{q}_{10}, \ldots, \tilde{q}_0), \tilde{q}),$$ wobei: $$(\tilde{q}_{10}, \ldots, \tilde{q}_0)_2 = (q_{10}, \ldots, q_0)_2 \underbrace{+}_{\mathrm{mod} 2^{12}} (0, 0, 0, e_7, \ldots, e_0)_2$$

Das hier beschriebene Schaltwerk ist in der Lage, zwischen zwei und acht 8-stellige Dualzahlen korrekt zu addieren. Um zu erreichen, daß das Schaltwerk nur sechs Zahlen addiert, kann man einen zusätzlichen binären Zähler einbauen, der die Eingaben zählt. Im Mealy-Automaten entspricht dies einer zusätzlichen Komponente in jedem Zustand, d. h. $Q = B^{11} \times B^3$. Endzustände sind dann alle Zustände, in denen die letzte Komponente gleich $(110)_2$ ist, und die Ausgabe ist immer dann leer, wenn der Zustand kein Endzustand ist.

Lösung zu Aufgabe 22

Zunächst noch einmal das Schaltwerk:



Der Mealy-Automat $A = (Q, \Sigma, \Delta, q_s, F, \delta)$ zur Polynom-Multiplikation über dem endlichen Körper $\mathbb{F}$ ist gegeben durch:

Tupelelement	**Idee**
$Q = \mathbb{F}^{n-k}$	Wir benötigen $(n-k)$ Register, in denen die zu Terme des Polynoms gespeichert werden.
$\Sigma = \mathbb{F}$	Die Eingabewerte geben den Koeffizienten des Eingabepolynoms an. Diese werden sukzessive in die Multiplikationskette geschoben, beginnend mit dem höchsten Koeffizienten.
$\Delta = \mathbb{F}$	Nach dem i-ten Schritt wird der Koeffizient von x^{n-i} des Ergebnispolynoms ausgegeben.
$q_s = (0, \ldots, 0)$	Zu Beginn sollen alle Delays gelöscht sein.
$F = \emptyset$	Wir benötigen keinen ausgezeichneten Zustand.
δ	$\delta : Q \times \Sigma \to Q \times \Delta, ((q_{n-k-1}, \ldots, q_0, e) \mapsto ((e, q_{n-k-1}, \ldots, q_1), a)$ Die Zustandsveränderung ist klar: Die Eingabe e kommt in das Delay q_{n-k-1}, während die Inhalte der anderen Delays um eine Stelle verschoben werden. Bei der Eingabe wird dabei mit dem höchsten Koeffizienten begonnen. Die Ausgabe a ergibt sich zu: $$a = e \cdot h_{n-k} \oplus \bigoplus_{i=0}^{n-k-1} q_i \cdot h_i$$ Dies ist unmittelbar aus dem Schaltwerk ersichtlich.

Auch hier kann man noch einen binären Zähler einfügen, der die eingegebenen Koeffizienten zählt (vgl. Aufg. 21).

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

8. Juni 1998

Musterlösungen zu Blatt 6

Lösung zu Aufgabe 23

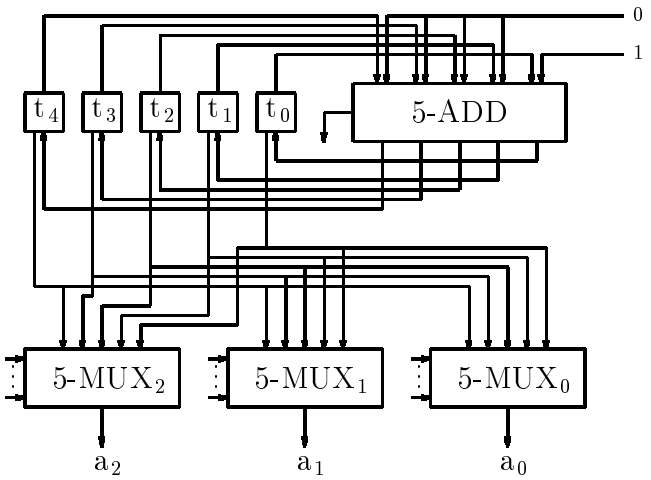
a) Diese Lösung verwendet einen Timer, der die verschiedenen Zeitpunkte bei der Lösung des WZK-Problems beschreibt. Die Zeitpunkte t werden folgendermaßen definiert:

t	**W**	**Z**	**K**	**B**	**Inhalt**	**Links**	**Rechts**	**Ausgabe**	**binär**
00000	1	1	1	1	$\emptyset$	wzk◯	$\emptyset$	L_Z	010
00001	1	1	1	1	z	wkⓩ	$\emptyset$	F	000
00010	1	0	1	0	z	wk	Ⓩ	E	111
00011	1	0	1	0	$\emptyset$	wk	◯z	F	000
00100	1	0	1	1	$\emptyset$	wk◯	z	L_W	100
00101	1	0	1	1	w	kⓦ	z	F	000
00110	0	0	1	0	w	k	ⓦz	E	111
00111	0	0	1	0	$\emptyset$	k	◯wz	L_Z	010
01000	0	0	1	0	z	k	Ⓩw	F	000
01001	0	1	1	1	z	kⓩ	w	E	111
01010	0	1	1	1	$\emptyset$	kz◯	w	L_K	001
01011	0	1	1	1	k	zⓚ	w	F	000
01100	0	1	0	0	k	z	ⓚw	E	111
01101	0	1	0	0	$\emptyset$	z	◯wk	F	000
01110	0	1	0	1	$\emptyset$	z◯	wk	L_Z	010
01111	0	1	0	1	z	Ⓩ	wk	F	000
10000	0	0	0	0	z	$\emptyset$	Ⓩwk	E	111

Der Mealy-Automat A = (Q, Σ, Δ, q_0, F, δ) kann definiert werden durch:

Tupelelement	Idee / Bemerkung
$\mathrm{Q} = B^5$	Wir benötigen 5 Delays, um den Zeitpunkt $t = (t_4, \ldots, t_0)$ zu speichern.
$\Sigma = \emptyset$	Es gibt keine Eingabe. Der Zustand ist durch den Zeitpunkt t gegeben.
$\Delta = B^3$	Es gibt fünf verschiedene Ausgaben; diese werden folgendermaßen definiert: $\begin{array}{c\|c\|c\|c\|c} L_W & L_Z & L_K & E & F \\ \hline 100 & 010 & 001 & 111 & 000 \end{array}$
$q_0 = (0,0,0,0,0)$	Der Automat startet zum Zeitpunkt $t = 0$.
$F = \{(1,0,0,0,1)\}$	Zu diesem Zeitpunkt ist das Problem gelöst.
δ	$\delta : Q' \to Q \times \Delta, q \mapsto (\tilde{q}, a),$ wobei $a = (a_2, a_1, a_0)$ wie in der Tabelle definiert sind, $Q' = \{q \in Q \mid (q)_2 \leq 16\}$, und $\tilde{q} = (\tilde{q_4}, \ldots, \tilde{q_0})$ für $q = (q_4, \ldots, q_0)$ gegeben ist durch: $(\tilde{q_4}, \ldots, \tilde{q_0})_2 = (q_4, \ldots, q_0)_2 + 1$

b) Die in a) beschriebene Lösung des WZK-Problems läßt sich wie folgt durch ein Schaltwerk realisieren. Dabei werden die Eingänge der drei 5-MUX-Bauteile entsprechend der Tabelle belegt. Die Anfangsbelegung der Register ist $t_0 = \cdots = t_4 = 0$.



Lösung zu Aufgabe 24

Mit der Bemerkung nach der Definition der Zweier-Komplementdarstellung ganzer Zahlen gilt:

$$\begin{aligned} (K_2(x))_2 &= (K_1(x))_2 + 1 \\ &= (2^n - 1) - x + 1 \\ &= 2^n - x \qquad (1) \end{aligned}$$

Es folgt:

$$\begin{aligned} K_2\big(((K_2(x))_2 + (K_2(y))_2) \bmod 2^n\big) &= K_2((2^n - x + 2^n - y) \bmod 2^n) \quad \text{nach (1)} \\ &= K_2((2^{n+1} - x - y) \bmod 2^n) \\ &= K_2((-x - y) \bmod 2^n) \end{aligned}$$

Lösung zu Aufgabe 25

Für die normalisierte Gleitkomma-Darstellung, in der der Exponent in Zweier-Komplement-Darstellung dargestellt wird, ergeben sich folgende Werte:

Darstellung:	**Gleitkomma**	**Festkomma**
größte Zahl	$2^{127} \cdot (1 - 2^{-23})$	$\frac{2^{31}-1}{2^8}$
kleinste Zahl	$2^{-128} \cdot 2^{-1} = 2^{-129}$	2^{-8}
größter Rundungsfehler (absolut)	2^{103}	2^{-9}
größter Rundungsfehler (relativ)	$\frac{1}{2^{-23}+1}$	1

Dabei berechnet sich der absolute Rundungsfehler der Zahl x als $|\mathrm{rd}(x) - x|$ und der relative Rundungsfehler als $\frac{|\mathrm{rd}(x)-x|}{x}$, wobei $\mathrm{rd}(x)$ die Zahl ist, auf die x gerundet wird. Der größte Rundungsfehler tritt in der Gleitkomma-Darstellung für $x = 2^{127} \cdot (2^{-1} + 2^{-24})$ und in der Festkomma-Darstellung für $x = 2^{-9}$ auf.

Lösung zu Aufgabe 26

Tupelelement	**Idee / Bemerkung**
$\mathrm{Q} = B^{4n}$	Es werden $4n$ Register benötigt, um den Zustand $q = (x, z) = (x_{2n-1}, \ldots, x_0, z_{2n-1}, \ldots, z_0)$ zu speichern: • Das *Schieberegister* x ist $2n$ Bits groß und wird in den höherwertigen Stellen von q kodiert. • Der *Akkumulator* z benötigt $2n$ Bits und soll durch die niederwertigen Stellen von q repräsentiert werden.
$\Sigma = B$	Die einzelnen Bits der Binärdarstellung von y werden sukzessive durch die Multiplikationsschlange geschoben.
$\Delta = B^{2n}$	Nach jeder Operation soll der Akkumulator ausgegeben werden. Dieser ist $2n$ Bits groß.
$q_0 = 0^n x_{n-1} \ldots x_0 0^{2n}$	Der Wert $x = (x_n, \ldots, x_0)_2$ muß oben mit n Nullen aufgefüllt werden. Der Akkumulator ist zu Beginn gelöscht ($2n$ Nullen).
$F = \emptyset$	Es gibt keinen ausgezeichneten Zustand.
δ	$\delta : Q \times \Sigma \to Q \times \Delta, (xz, e) \mapsto (\tilde{x}\tilde{z}, \tilde{z})$, wobei: $x = (x_{2n-1}, \ldots, x_0)$ $z = (z_{2n-1}, \ldots, z_0)$ $\tilde{x} = (x_{2n-2}, \ldots, x_0, x_{2n-1})$ $\tilde{z} = (z_{2n-1}, \ldots, z_0)_2 + e \cdot (x_{2n-1}, \ldots, x_0)_2 \pmod{2^{2n}}$

Lösung zu Aufgabe 27

a) Die Idee hierbei ist es, jeweils *drei* Schritte auf einmal zu bearbeiten. Das wird dadurch erreicht, daß bei jedem Schritt drei Stellen der Eingabezahl y eingelesen werden. Diese Stellen y_2, y_1 und y_0 werden nun mit der Zahl x (Schieberegister) multipliziert und dann entsprechend addiert. Hier bei bedeutet „entsprechend“, daß das Ergebnis $y_2 \cdot x$ um zwei Stellen und das Ergebnis $y_1 \cdot x$ eine Stelle nach links geschoben in den CSA eingegeben wird. Hieraus folgt, daß eine Sonderregelung für das Schieberegister nötig ist, das nach jedem Schritt um *drei* Stellen rotieren muß. Die CSA's führen schnelle Zwischenadditionen durch, bevor die Werte mit einem „normalen“ Addierer bearbeitet werden. Das Schaltwerk ist in Abbikdung 1 gezeigt.

b) Das Schaltwerk aus a) kann durch folgenden Mealy-Automaten beschrieben werden:

Tupelelement	**Idee / Bemerkung**
$\mathrm{Q} = B^{28}$	Es werden 28 Register benötigt, um den Zustand $q = (x, z) = (x_{13}, \ldots, x_0, z_{13}, \ldots, z_0)$ zu speichern. Dabei soll x wieder den Inhalt des Schieberegisters beinhalten und z den Zustand des Akkumulators repräsentieren.
$\Sigma = B^3$	Bei diesem Schaltwerk werden in jedem Schritt drei Stellen von y eingegeben.
$\Delta = B^{14}$	Der Akkumulator ist 14 Bit groß.
$q_0 = 0^7 x_7 \ldots x_0 0^{14}$	Der Wert $x = (x_7, \ldots, x_0)_2$ muß oben mit 7 Nullen aufgefüllt werden. Der Akkumulator ist zu Beginn gelöscht (14 Nullen).
$F = \emptyset$	Es gibt keinen ausgezeichneten Zustand.
δ	$\delta : Q \times \Sigma \to Q \times \Delta, (xz, e) \mapsto (\tilde{x}\tilde{z}, \tilde{z})$, wobei: $\begin{aligned} x &= (x_{13}, \ldots, x_0) \\ z &= (z_{13}, \ldots, z_0) \\ e &= (e_2, e_1, e_0) \\ \tilde{x} &= (x_{10}, \ldots, x_3, x_{13}, x_{12}, x_{11}) \\ (\tilde{z})_2 &= (z_{13}, \ldots, z_0)_2 \\ &\quad + e_2 \cdot (x_{11}, \ldots, x_0, 0, 0)_2 \\ &\quad + e_1 \cdot (x_{12}, \ldots, x_0, 0)_2 \\ &\quad + e_0 \cdot (x_{13}, \ldots, x_0)_2 \qquad (\mathrm{mod}\ 2^{14}) \end{aligned}$

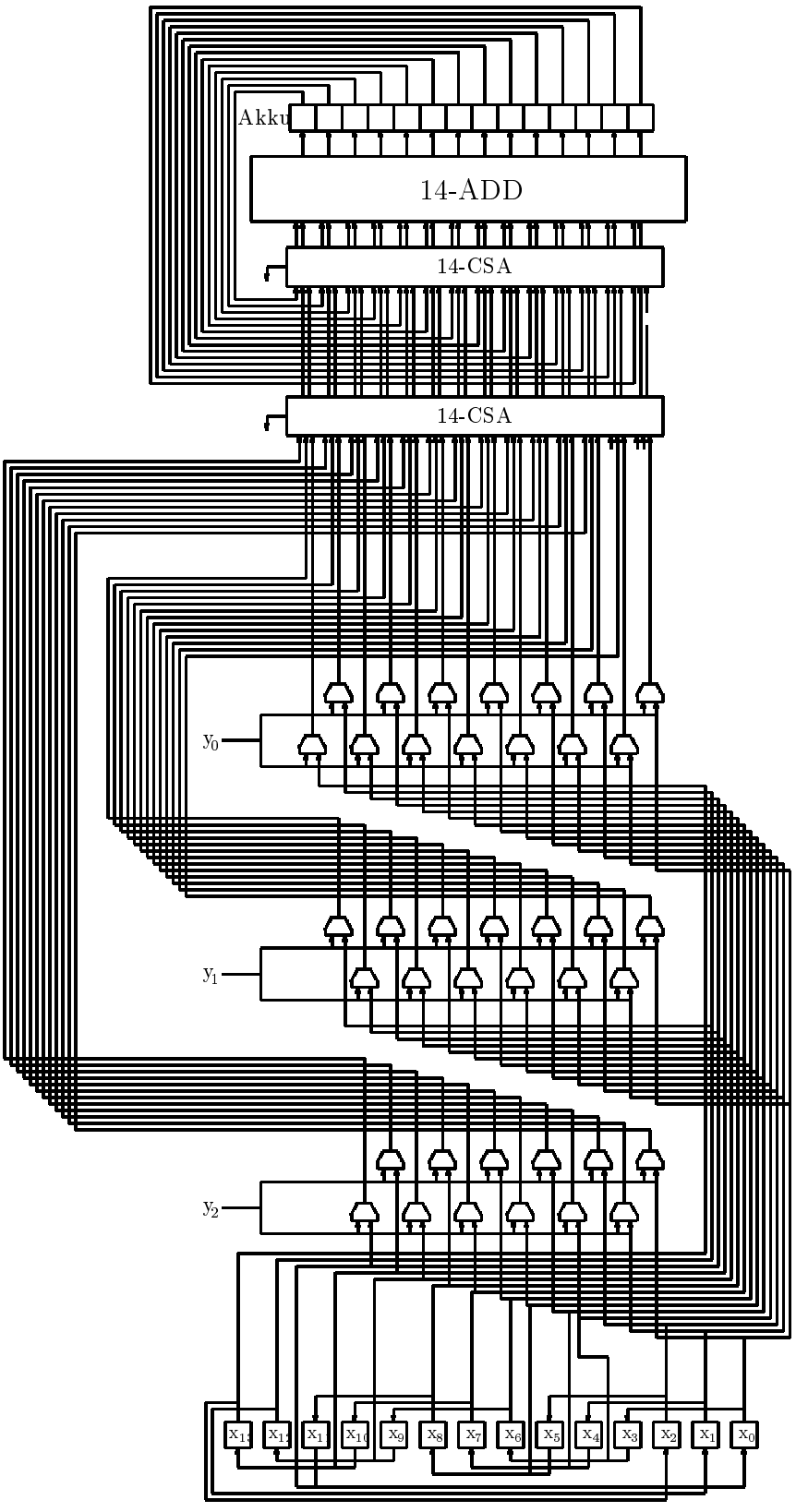


Abbildung 1. Das Schaltwerk zu Aufgabe 27 a)

Lösung zu Aufgabe 28

Wir zeigen, daß die bisher verwendeten Addierwerke zur Addition von Zahlen aus $\mathbb{N}_0$ auch für Zahlen im Zweierkomplement verwendet werden können, d.h. es sind keine zusätzlichen Mechanismen zur Addition von Zahlen im Zweierkomplement nötig.
Seien also x, y ganze Zahlen in Zweierkomplement-Darstellung.
Wir führen folgende Fallunterscheidungen durch:

- $x, y \geq 0$:
  Dieser Fall ist trivial.

- $x \geq 0$ und $y < 0$:
  In diesem Fall wird y dargestellt als $K_2(-y)$. Aus der Vorlesung (Lemma) ist bekannt: $a - b = a + (K_2(b))_2 \pmod{2^n}$. Damit folgt: $x + K_2(-y) = x - (-y) = x + y$.

- $x < 0$ und $y \geq 0$:
  Dieser Fall ist aufgrund der Kommutativität der verwendeten Operationen in einem Addierwerk analog zum zweiten Fall.

- $x < 0$ und $y < 0$:
  Hier werden die Zahlen x und y durch $K_2(-x)$ und $K_2(-y)$ dargestellt. Mit dem oben zitierten Lemma aus der Vorlesung folgt: $K_2(-x)+K_2(-y) = K_2(-x)-(-y) = K_2(-x) + y = y - (-x) = x + y$.

Dies bedeutet, daß die bisher entwickelten Additionswerke weiterhin genutzt werden können. Dies gilt auch für die Verfahren zur Beschleunigung der Addition.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

15. Juni 1998

Musterlösungen zu Blatt 7

Lösung zu Aufgabe 29

a) Hier besteht das Programm aus drei Vergleichs- / Vertauschblöcken:

- Der erste Block besteht aus den Anweisungen 1–8. Dabei werden die Register R_1 und R_2 verglichen und gegebenenfalls vertauscht.
- Der zweite Block beginnt bei Anweisung 9 und endet mit Anweisung 17. Hier werden die Register R_2 und R_3 sortiert.
- Der dritte und letzte Block geht von Zeile 18 bis Zeile 26, vergleicht das Register R_1 mit dem (eventuell neu belegten) Register R_2 und sortiert diese gegebenenfalls.

Es ist klar, daß durch diese Vergleiche (und Vertauschungen) die drei Zahlen in der gewünschten Reihenfolge sortiert werden.

Adresse	Befehl	Operand	Kommentar
0	LoadR	1	Lade den Wert von R_1 in den Akkumulator.
1	SubR	2	Subtrahiere den Wert von R_2.
2	GotoLe	9	Ist R_2 größer als R_1 dann weiter...
3	LoadR	1	Sonst lade den Wert von R_1 in den Akkumulator.
4	StoreR	0	Und speichere diesen in R_0.
5	LoadR	2	Lade den Wert aus R_2...
6	StoreR	1	...und schiebe ihn nach R_1.
7	LoadR	0	Lade den ursprünglichen Inhalt aus R_1...
8	StoreR	2	...und schreibe diesen nach R_2. Jetzt ist $\delta(R_2) \geq \delta(R_1)$.
9	LoadR	2	Lade den Wert von R_2 in den Akkumulator.
10	SubR	3	Subtrahiere den Wert von R_3.
11	GotoLe	27	Ist R_3 größer als R_2 dann sind die Werte richtig sortiert.
12	LoadR	2	Sonst lade den Wert von R_2 in den Akkumulator.
13	StoreR	0	Und speichere diesen in R_0.
14	LoadR	3	Lade den Wert aus R_3...
15	StoreR	2	...und schiebe ihn nach R_2.
16	LoadR	0	Lade den ursprünglichen Inhalt aus R_2...
17	StoreR	3	...und schreibe diesen nach R_3.
18	LoadR	1	Lade den Wert von R_1 in den Akkumulator.
19	SubR	2	Subtrahiere den Wert von R_2.
20	GotoLe	27	Ist R_2 größer als R_1 dann sind die Werte sortiert.
21	LoadR	1	Sonst lade den Wert von R_1 in den Akkumulator.
22	StoreR	0	Und speichere diesen in R_0.
23	LoadR	2	Lade den Wert aus R_2...
24	StoreR	1	...und schiebe ihn nach R_1.
25	LoadR	0	Lade den ursprünglichen Inhalt aus R_1...
26	StoreR	2	...und schreibe diesen nach R_2.
27	Stop		

b) Die Idee hier ist es, zwei Zeiger zu verwenden. Der eine Zeiger steht im Register R_0 und zeigt auf eine Adresse in der unteren Hälfte des Arrays. Der andere Zeiger ist im Register R_1 gespeichert und adressiert die (an der Arraymitte gespiegelte) Adresse $101 - R_0$. Nun werden die Inhalte der entsprechenden Speicherzellen vertauscht und die Zeiger entsprechend verändert. Dabei beginnt das Programm bei der Mitte des Arrays und arbeitet sich bis an die Arraygrenzen vor, d.h. daß R_0 dekrementiert und R_1 inkrementiert wird.

Adresse	**Befehl**	**Operand**	**Kommentar**
0	Load	51	Es werden jeweils die Speicherzellen M_{R_0}...
1	StoreR	1	...und M_{R_1} vertauscht. Dazu müssen...
2	Load	50	...die entsprechenden Adressen geladen werden.
3	StoreR	0	Speichere die linke Adresse.
4	LoadIR	0	Lade die Speicherzelle M_{R_0} in den Akkumulator.
5	StoreR	2	Speichere den Wert temporär in R_2.
6	LoadIR	1	Lade die Speicherstelle M_{R_1} in den Akkumulator.
7	StoreIR	0	Lege in Inhalt des Akkumulators in der anderen Speicherzelle ab.
8	LoadR	2	Hole den temporär gespeicherten Wert aus R_2...
9	StoreIR	1	...und schreibe in an die andere Adresse.
10	LoadR	1	Lade die Adresse auf der rechten Seite.
11	Add	1	Und gehe zur nächsten Adresse.
12	StoreR	1	Speichere den neuen Wert.
13	LoadR	0	Lade die Adresse auf der linken Seite.
14	Sub	1	Und gehe zur nächsten Adresse.
15	GotoGt	3	Wenn nicht 0, dann weitermachen.
16	Stop		Fertig.

Lösung zu Aufgabe 30

Hier wird das bisher berechnete Minimum bzw. Maximum in dem Register R_1 bzw. R_2 gespeichert und das Feld von der Zelle M_{100} ab durchsucht. In jedem Schritt wird der Inhalt der Speicherzelle M_{R_0} mit dem Wert von R_1 und R_2 verglichen und dann gegebenenfalls übernommen. Nach Beendigung der Suche, d.h. nach Erreichen der Adresse 0, werden die Werte in die entsprechenden Speicherplätze M_{101} und M_{102} übertragen.

Adresse	Befehl	Operand	Kommentar
0	Load	100	Lade die Adresse der letzten Zahlenadresse.
1	StoreR	0	Speichere sie im Register R_0, das als Zeiger dient.
2	LoadIR	0	Lade die Zahl in M_{100} in den Akkumulator.
3	StoreR	1	Diese Zahl ist Startminimum...
4	StoreR	2	... und Startmaximum.
5	LoadR	0	Lade die Adresse der letzten bearbeiteten Zahl.
6	Sub	1	Gehe zur neuen Adresse.
7	GotoEq	20	Wenn bei Adresse 0 angelangt, beenden.
8	StoreR	0	Anderenfalls speichere die neue Adresse.
9	LoadIR	0	Lade die Zahl an der neuen Adresse in den Akkumulator.
10	SubR	1	Subtrahiere das bisherige Minimum.
11	GotoGt	14	Ist das Ergebnis größer als 0 ? (Sonst neues Minimum)
12	LoadIR	0	Lade die Zahl erneut.
13	StoreR	1	Und speichere diese im Minimumregister M_1.
14	LoadIR	0	Lade die Zahl an der neuen Adresse in den Akkumulator.
15	SubR	2	Subtrahiere das bisherige Maximum.
16	GotoLe	5	Ist das Ergebnis kleiner als 0 ? (Sonst neues Maximum)
17	LoadIR	0	Lade die Zahl erneut.
18	StoreR	2	Und speichere diese im Maximumregister M_2.
19	Goto	5	Sonst wiederhole den Vorgang mit der nächsten Adresse.
20	LoadR	1	Lade das Minimum in den Akkumulator...
21	StoreM	101	...und speichere es an Adresse M_{101}.
22	LoadR	2	Lade das Maximum in den Akkumulator...
23	StoreM	102	...und speichere es an der Adresse M_{102}.
24	Stop		Das war es...

Lösung zu Aufgabe 31

In diesem Programm wird der Zähler n, der im Register R_0 enthalten ist, dekrementiert, bis er den Wert 0 erreicht hat. In jedem Schritt wird aus den Werten $F(i-2)$ und $F(i-1)$, die im Register R_1 bzw. R_2 enthalten sind, der Wert $F(i)$ berechnet und in das Register R_2 eingetragen, danach wird der Inhalt von R_2 in das Register R_1 geschrieben. Nach n Schritten befindet sich somit das gewünschte Ergebnis im Register R_1.

Adresse	**Befehl**	**Operand**	**Kommentar**
0	Load	1	Lade den Wert 1 in den Akkumulator.
1	StoreR	1	Register R_1 enthält nun den Wert $F(1) = 1$.
2	StoreR	2	Register R_2 enthält nun den Wert $F(2) = 1$.
3	LoadR	0	Lade n in den Akkumulator.
4	Add	-1	Subtrahiere 1 von der Anzahl der noch durchzuführenden Schritte.
5	GotoEq	13	Wenn nur noch 0 Schritte durchzuführen sind, dann beende das Programm.
6	StoreR	0	Anderenfalls speichere die Anzahl der verbleibenden Schritte.
7	LoadR	2	Lade den Wert von $F(i-1)$ in den Akkumulator.
8	AddR	1	Addiere den Wert von $F(i-2)$ um $F(i)$ zu berechnen.
9	StoreR	2	Lege $F(i)$ in R_2 ab.
10	SubR	1	Subtrahiere $F(i-2)$ um $F(i-1)$ zu erhalten.
11	StoreR	1	Und lege den Wert $F(i-1)$ in R_1 ab.
12	Goto	3	Wiederholen.
13	Stop		Beende das Programm. Das Ergebnis steht in R_1.

Dieses Programm benötigt nur drei Register zur Ermittlung der n-ten Fibonacci-Zahl.

Lösung zu Aufgabe 32

Es gilt: **Ein Von-Neumann-Rechner mit endlichen Speicher und selbstmodifizierendem Programm kann durch einen Mealy-Automaten beschrieben werden.**

Beweis: In einem Von-Neumann-Rechner R existiert nur eine begrenzte Anzahl von Registern, wie etwa der *Akkumulator A*, ein *Multiplikationsregister MR*, ein *Linkregister L* sowie ein *Pufferregister MBR* (vgl. Oberschelp/Vossen: *Rechneraufbau und Rechnerstrukturen*). Zusätzliche Register werden benötigt, um etwa die aktuell auszuführende Position im Assemblerprogramm zu speichern. Seien also insgesamt k Register verfügbar. Diese Register können jeweils (da dies sonst der Annahme des endlichen Speichers widersprechen würde) nur endlich viele verschiedene Werte aufnehmen. O.B.d.A. gehen wir davon aus, daß es sich hierbei um natürliche Zahlen aus $\Gamma = \{0, \ldots, n_1\}$ handelt. Der endliche RAM-Speicher enthält sowohl das Programm als auch die Daten. Er besteht (da endlich) aus m Speicherzellen, die jeweils Werte aus einem endlichen Symbolalphabet beinhalten. Dieses Symbolalphabet sei o.B.d.A. durch $\Omega = \{0, \ldots, n_2\}$ gegeben. Hieraus folgt, daß wir den kompletten Zustand des Von-Neumann-Rechners R durch ein Tupel der Form $(x_1, \ldots, x_k, y_1, \ldots, y_m) \in \Gamma^k \times \Omega^m$ modellieren können. Als Eingabealphabet Σ wählen wir die leere Menge. Auch auf Ausgaben können wir verzichten und setzen $\Delta = \emptyset$. Unser Anfangszustand q_0 ist durch die anfängliche Registerbelegung in den x_i und die im Speicher enthaltenen Instruktionen bzw. Daten in y_i gegeben. Auf einen ausgezeichneten Zustand verzichten wir und setzen: $F = \emptyset$. Unsere Übergangsfunktion δ bildet nun in jedem Schritt den Zustand, d.h. die Registerinhalte sowie den Speicher auf die Register- und Speicherbelegungen nach der Abarbeitung der aktuellen Instruktion ab. Befindet sich die Maschine beispielsweise in Zustand $q = (x_1, \ldots, x_k, y_1, \ldots, y_m)$ und wird der Befehl `StoreM 1` ausgeführt (der Akkumulator enthalte den Wert 7), so gilt

$$\delta\big((x_1, \ldots, x_k, y_1, \ldots, y_m)\big) = (x'_1, \ldots, x'_k, y'_1, \ldots, y'_m),$$

wobei der Instruktionszeiger (beispielsweise durch x_5 gegeben) eine Stelle weiterrückt ($x'_5 = x_5 + 1$), die Speicherzelle 1 den Wert 7 enthält ($y'_1 = 7$) und alle anderen Komponenten unverändert bleiben ($x'_i = x_i$ und $y'_j = y_j$ für alle $i \in \{1, \ldots, k\} \setminus \{5\}$ und $j \in \{2, \ldots, m\}$). In analoger Weise können sämtliche Assemblerbefehle in eine geeignete δ-Abbildung überführt werden.
Daraus folgt die Behauptung.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

22. Juni 1998

Musterlösungen zu Blatt 8

Lösung zu Aufgabe 33

a) Nach der ersten Ausführung von Zeile 9 liegen folgende Belegungen vor: (siehe auch kommentiertes Programm in Aufgabenteil c))

- **Register:** Wie man leicht sieht, wird lediglich das Register R_0 in den Zeilen 1 bis 9 benutzt. Nach dem Schleifendurchlauf (Zeile 2 bis Zeile 7) wird in Zeile 8 mit `Load 2` der Wert 2 in den Akkumulator geladen und dann (Zeile 9) in das Register R_0 geschoben (`StoreR 0`).

- **Speicher:** In den Zeilen 1 und 2 wird der Inhalt der Speicherzelle M_0 in das Register R_0 transferiert. In die dadurch adressierte Speicherzelle $M_{\delta(R_0)}$ wird eine 1 geschrieben (Zeile 3,4). R_0 wird nun heruntergezählt (Zeile 6: `Sub 1`), bis durch das Subtrahieren der Wert 0 erreicht wird. Dabei wird jedesmal eine 1 in die entsprechende Speicherzelle geschrieben. Man beachte jedoch, daß das Dekrementieren *nach* dem Speicherzugriff stattfindet, so daß kein Zugriff auf die Speicherstelle M_0 vorgenommen wird.

R_0	M_0	M_1	M_2	...	M_{n-1}	M_n	M_{n+1}	M_{n+2}
2	n	1	1	...	1	1	?	...

b) Das Programm enthält zwei Fehler:

- **Fehler 1:** *Beim Löschen der Vielfachen der Primzahl p wird diese selber gelöscht.*
  Das Programm beginnt beim Löschen der Vielfachen der Primzahl $p = R_0$ mit der Zahl $\left\lfloor \frac{i}{p} \right\rfloor \cdot p$, d.h. mit dem größten Vielfachen von p, das kleiner ist als M_0. Die Abbruchbedingung im Zeile 19 (`GotoGt 13`) führt erst dann zu einem Abbruch, wenn durch das Subtrahieren die Zahl 0 erreicht wird, verzweigt also auch dann zur Zeile 13, wenn der Akkumulator den Wert p enthält. In die Speicherzelle $M_p = M_{\delta(R_1)} = M_{\delta(R_0)}$ wird nun nach dem Verzweigen die Zahl 0 eingetragen.
  Lösung: *Nach Beendigung der Schleife Zeile 13 bis Zeile 19 muß in die Speicherzelle $M_p = M_{\delta(R_0)}$ der Wert 1 geschrieben werden.*

- **Fehler 2:** *Die gefundene nächsthöhere Primzahl in R_0 wird überschrieben.*
  Nach dem Markieren der Nichtprimzahlen (Vielfache von p) muß die nächsthöhere Primzahl $\tilde{p}$ gesucht werden. Dies geschieht in den Zeilen 20 bis 27 des ursprünglichen

Programms. Dort werden die Speicherzellen M_i $(i = p+1, p+2, \dots)$ solange nach einer Eins durchsucht, bis entweder eine neue Primzahl $\tilde{p}$ gefunden wurde oder aber über die Zahl $n = \delta(M_0)$ (die betrachtet werden soll) hinausgelaufen wurde. Die Abfrage, ob die Speicherzelle $M_{\delta(R_0)}$ eine 1 enthält, geschieht in den Zeilen 25 und 26 (`LoadIR 0, GotoGt 9`). In der Zeile 25 wird der Inhalt der Speicherzelle $M_{\delta(R_0)}$ in den Akkumulator geladen. In Zeile 26 erfolgt der Sprung, wenn der Akkumulator nun eine 1 enthält. In diesem Fall enthält das Register R_0 die neue Primzahl $\tilde{p}$. Nun geschieht der Fehler: Beim Sprung in die Zeile 9 wird der Inhalt von R_0 durch den Akkumulator überschrieben, d.h. R_0 enthält nun eine 1.
Lösung: *Ersetze Anweisung 26 (bzw. 28) durch* `GotoGt 10`.

c) Hier also das korrekte kommentierte Programm:

Adr.	Befehl	Op.	Kommentar
1:	`LoadM`	0	Lade die Zahl n aus M_0 in den Akkumulator.
2:	`StoreR`	0	Speichere sie im Register R_0.
3:	`Load`	1	Lade eine 1 in den Akkumulator.
4:	`StoreIR`	0	Schreibe die 1 an die Speicherstelle $M_{\delta(R_0)} = M_n$.
5:	`LoadR`	0	Lade die Adresse in den Akkumulator...
6:	`Sub`	1	und erniedrige die Adresse um 1.
7:	`GotoGt`	2	Wiederhole, bis die Speicherstellen $M_n, \dots, M_1$ mit Einsen belegt sind.
8:	`Load`	2	Lade die Startprimzahl $p = 2$ in den Akkumulator...
9:	`StoreR`	0	...und speichere sie als Schrittweite in R_0.
10:	`LoadM`	0	Lade die Zahl $n = \delta(M_0)$ in den Akkumulator.
11:	`DivR`	0	Teile sie durch die Primzahl p (runde ab)...
12:	`MulR`	0	... und multipliziere sie mit p. (D.h. bilde: $\left\lfloor \frac{M_0}{p} \right\rfloor \cdot p$).
13:	`StoreR`	1	Speichere die aktuelle Position im Register R_1.
14:	`Load`	0	Lade die Zahl 0 in den Akkumulator.
15:	`StoreM`	1	Die Zahl 1 ist keine Primzahl, also M_1 löschen.
16:	`StoreIR`	1	R_1 enthält ein Vielfaches von p, also $M_{\delta(R_1)}$ mit 0 belegen.
17:	`LoadR`	1	Lade die Position in den Akkumulator...
18:	`SubR`	0	... und subtrahiere p um zum nächstkleineren Vielfachen zu gelangen.
19:	`GotoGt`	13	Wiederhole, wenn noch nicht alle Vielfachen gelöscht.
20:	`Load`	1	Lade eine 1 in den Akkumulator.
21:	`StoreIR`	0	Die Zahl p in R_0 ist Primzahl, $M_{\delta(R_0)}$ wurde aber gelöscht (siehe b)). Also wieder eine 1 reinschreiben.
22:	`LoadR`	0	Lade die letzte Schrittweite in den Akkumulator...
23:	`Add`	1	... und erhöhe sie um eins.
24:	`StoreR`	0	Speichere die neue Schrittweite in R_0.
25:	`SubM`	0	Subtrahiere die Zahl n.
26:	`GotoGt`	30	Wenn die Schrittweite größer ist als n, dann ist das Programm fertig.
27:	`LoadIR`	0	Lade den Inhalt von $M_{\delta(R_0)}$ in den Akkumulator.
28:	`GotoGt`	10	Ist dieser = 1, so ist die neue Schrittweite eine Primzahl. Filtere also deren Vielfache raus.
29:	`Goto`	22	Anderenfalls versuche es mit der nächsten Schrittweite.
30:	`Stop`		Das war es...

Lösung zu Aufgabe 34

Der Trick liegt darin, daß man s_i induktiv berechnen kann als:

$$s_i = \begin{cases} 0 & \text{falls } i = 0 \\ s_{i-1} + a_i & \text{falls } 1 \leq i \leq 10 \\ s_{i-1} + a_i - a_{i-10} & \text{sonst} \end{cases}$$

Im folgenden Programm enthält das Register R_0 den Wert s_i, während R_1 den Wert i selber, R_2 die Zahl $i - 10$ und R_3 die Zahl $i + 100$ enthält.

Adr.	Befehl	Op.	Kommentar
1:	`Copy`	`R0,0`	Lade den Wert $s_0 = 0$ in das Register R_0.
2:	`Copy`	`R1,1`	Lade die aktuelle Adresse $i = 1$ nach R_1.
3:	`Copy`	`R2,-9`	Lade den Wert $i - 10 = \delta(R_1) - 10 = -9$ nach R_2.
4:	`Copy`	`R3,101`	Lade den Wert $i + 100 = \delta(R_1) + 100 = 101$ nach R_3.
5:	`LoadIR`	`1`	Lade den Wert a_i in den Akkumulator...
6:	`Add`	`R0,R0`	...und addiere ihn auf R_0, das nun den Wert $s_{i-1} + a_i$ enthält.
7:	`LoadR`	`2`	Lade den Wert $i - 10$ in den Akkumulator.
8:	`GotoGt`	`11`	Wenn $(i - 10) > 0$, also $i > 10$, dann subtrahiere a_{i-10}.
9:	`LoadR`	`0`	Sonst lade den Wert s_i in den Akkumulator.
10:	`Goto`	`13`	Und springe zur Verarbeitung des Wertes.
11:	`LoadIR`	`2`	Lade den Wert 0 in den Akkumulator.
12:	`Sub`	α`,R0`	Berechne den Wert $-a_{i-10}$.
13:	`StoreIR`	`3`	Speichere s_i in Adresse M_{i+100}.
14:	`Load`	`1`	Lade die Zahl 1 zum Inkrementieren in den Akkumulator.
15:	`Add`	`R1,R1`	Gehe zur neuen Adresse $(i := i + 1)$.
16:	`Add`	`R2,R2`	Berechne das neue $i - 10$.
17:	`Add`	`R3,R3`	Berechne das neue $i + 100$.
18:	`LoadR`	`1`	Lade die neue Adresse in den Akkumulator.
19:	`Sub`	`100`	Subtrahiere 100, d.h. Akku enthält 1, wenn $i = 101$.
20:	`GotoLe`	`5`	Wenn nicht, wiederhole.
21:	`Stop`		

Das Programm benötigt 100 (Zeile 6) + 100 (Zeile 15) + 100 (Zeile 16) + 100 (Zeile 17) = 400 Additionen sowie 90 (Zeile 12) + 100 (Zeile 19) = 190 Subtraktionen.

Lösung zu Aufgabe 35

a) $\delta(\alpha) := \delta(M_{\delta(R_0)})$.

Befehl	Op.	Mikroprogramm
`LoadIR`	`0`	1) Schalte w_0, rm. 2) Schalte $mbrmar$. 3) Schalte $mbrin$. 4) Schalte wm, ra.

b) $\delta(\alpha) := \max(\delta(R_0), \delta(R_1))$.

Befehl	Op.	Mikroprogramm
`LoadR`	`0`	1) Schalte w_0, ra.
`SubR`	`1`	1) Schalte w_1. 2) Anweisung ALU: Subtrahiere. 3) Schalte akk.

Falls Akku kleiner Null		
Befehl	**Op.**	**Mikroprogramm**
`LoadR`	`0`	1) Schalte w_0, ra.

Sonst		
Befehl	**Op.**	**Mikroprogramm**
`LoadR`	`1`	1) Schalte w_1, ra.

c) $\delta(\alpha) := \delta(M_{\delta(R_0)\cdot 4+\delta(R_1)})$.

Befehl	Op.	Mikroprogramm
`LoadR`	`0`	1) Schalte w_0, ra.
`Mul`	`4`	1) Anweisung ALU: Multipliziere.
`AddR`	`1`	1) Schalte w_1. 2) Anweisung ALU: Addiere. 3) Schalte akk.
`StoreR`	`2`	1) Schalte wa,r_2.
`LoadIR`	`2`	1) Schalte w_2, rm. 2) Schalte $mbrmar$. 3) Schalte $mbrin$. 4) Schalte wm, ra.

Lösung zu Aufgabe 36

Die Idee hier ist es, im Register R_1 die Anzahl der aufeinanderfolgenden Nullen zu zählen und dieses Register dann bei der 1-er Folge wieder zu dekrementieren. Dadurch wird ein einfaches Kriterium ($R_1 = 0$) für den Programmabbruch gewonnen.
Zunächst sucht das Programm nach der ersten 1 (Zeile 2 bis 5) und setzt das Register R_1 dann auf -1. Dadurch ist garantiert, daß durch das Dekrementieren R_1 niemals den Wert 0 annimmt und diese Programmschleife also bis zur nächsten 0 durchlaufen wird. Jetzt wird der Zähler R_1 auf Null gestetzt (Zeile 16) und die aufeinanderfolgenden Nullen werden gezählt (Zeile 17-23). Sobald eine 1 gelesen wird, springt das Programm in die Programmschleife, die die 1en bearbeitet und dabei R_1 herunterzählt.

Adr.	Befehl	Op.	Kommentar
1:	`Copy`	`R0,0`	Wir beginnen bei Adresse 1 (da R_0 sofort inkrementiert wird, setze also: $R_0 := 0$).
2:	`Load`	`1`	Lade eine 1 in den Akkumulator zum Inkrementieren...
3:	`Add`	`R0,R0`	... und erhöhe die Position um eins.
4:	`LoadIR`	`0`	Lade das Zeichen, das in dieser Position steht.
5:	`GotoEq`	`2`	Wenn es eine 0 ist, dann lese die nächste Stelle.
6:	`Copy`	`R1,-1`	Ansonsten setzte die Anzahl vorläufig auf -1, um die Einsen zu überlesen.
7:	`Load`	`1`	Lade eine 1 in den Akkumulator zum Inkrementieren.
8:	`Add`	`R0,R0`	... und erhöhe die Position um eins.
9:	`Load`	`-1`	Lade eine -1 in den Akkumulator zum Dekrementieren.
10:	`Add`	`R1,R1`	... und erniedrige die Anzahl um eins.
11:	`Load`	`R1`	Vergleiche die Anzahl mit 0.
12:	`GotoEq`	`25`	Wenn ja, dann wurde das i gefunden.
13:	`LoadIR`	`0`	Lade das Zeichen, das auf der aktuellen Position $M_{\delta(R_0)}$ steht.
14:	`GotoEq`	`16`	Wenn dort eine 0 steht, dann verlasse die Schleife.
15:	`Goto`	`7`	Ansonsten lese weitere Einsen.
16:	`Copy`	`R1,0`	Setze die Anzahl der gelesenen Nullen auf 0.
17:	`Load`	`-1`	Lade eine -1 in den Akkumulator zum Dekrementieren.
18:	`Add`	`R2,R0`	Und setze R_2 auf das potentielle i, das sich eine Stelle *vor* der ersten Null befindet.
19:	`Load`	`1`	Lade eine 1 in den Akkumulator zum Inkrementieren.
20:	`Add`	`R1,R1`	... und erhöhe die Anzahl um eins.
21:	`Add`	`R0,R0`	... und erhöhe die Position um eins.
22:	`LoadIR`	`0`	Lade das Zeichen, das in dieser Position steht.
23:	`GotoEq`	`19`	Wenn es eine Null ist, dann lese die nächste Stelle.
24:	`Goto`	`7`	Anderenfalls bestimme die Anzahl der Einsen.
25:	`Stop`		Die Zahl i steht im Register R_2.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

29. Juni 1998

Musterlösungen zu Blatt 9

Lösung zu Aufgabe 37

Für dieses Programm wird erwartet, daß die Parameter folgendermaßen übergeben werden:

- Das Register R_1 enthält die Höhe k der Matrix A.
- In R_2 befindet sich die Breite ℓ der Matrix A.
- Die Höhe k' der Matrix B wird im Register R_3 übergeben.
- Die Breite ℓ' dieser Matrix ist durch R_4 gegeben.

Dies kann offensichtlich durch vier `Copy`-Befehle erreicht werden. Anstatt von festen Positionen der Matrizen werden diese ebenfalls als Parameter übergeben.

- Die Adresse der Matrix A ist im Register R_5 gespeichert.
- Die Position von B wird mit R_6 übergeben.

Die festen Adressen für die Matrix A (4) und Matrix B ($k \cdot \ell + 4$) können durch folgende Assembleranweisungen bestimmt werden:

Adr.	Befehl	Op.	Kommentar
1:	`Copy`	`R5,4`	Die Matrix A befindet sich stets an Adresse 4.
2:	`Mul`	`R6,R1,R2`	Die Adresse von B berechnet sich zu: $k \cdot \ell + 4$.
3:	`Add`	`R6,R6,R5`	

Da Multiplikationen bei den meisten Prozessoren teuerer sind als Additionen wird das Assemblerprogramm dahingehend optimiert, daß keine Multiplikationen benutzt werden. Dazu werden die neuen Adressen bei Zeilensprüngen stets durch Offsetverschiebungen und nicht durch Multiplikationen berechnet.

Adr.	Befehl	Op.	Kommentar
1:	`Sub`	`R1,R1,R3`	Maximale y-Position bestimmen, in der Matrix B in A liegen kann. Dieser Wert dient auch gleichzeitig als Offsetverschiebung.
2:	`Copy`	`R7,R5`	Dies ist die Adresse (!), in der die Suche in A startet.
3:	`Add`	`R8,R2,-1`	Dies ist die Anzahl, die in A noch in x-Richtung zu bearbeiten sind.
4:	`Sub`	`R8,R8,R4`	Davon muß natürlich die Breite der Matrix B subtrahiert werden.
5:	`Add`	`R9,R1,-1`	Dies ist die Anzahl der Stellen, die A noch in y-Richtung zu bearbeiten ist.
6:	`Copy`	`R10,R7`	Dies ist die Startposition in Matrix A, von der ab an A durchsucht wird.
7:	`Copy`	`R11,R6`	Die Matrix B wird von links oben durchsucht und jeweils mit A verglichen.
8:	`Copy`	`R13,R4`	Es müssen ℓ' Werte in x-Richtung verglichen werden...
9:	`Copy`	`R12,R3`	..und k' Werte in y-Richtung, um die gesamte Matrix B auf A zu legen.
10:	`Sub`	`R14,[R10],[R11]`	Lade den Wert aus A in den Akkumulator und subtrahiere den Wert aus Matrix B.
11:	`GotoEq`	`13,R14`	Weitermachen, wenn $= 0$ also Werte gleich sind.
12:	`Goto`	`21`	Sonst mit nächster Position versuchen.
13:	`Add`	`R10,R10,1`	Nächste Position in A.
14:	`Add`	`R11,R11,1`	Nächste Position in B.
15:	`Sub`	`R12,R12,1`	Es wurde eine weitere Stelle bearbeitet.
16:	`GotoGt`	`10,R12`	Wenn die Spalte noch nicht bearbeitet wurde, dann nächste Position.
17:	`Add`	`R10,R10,R1`	Ansonsten gehe mit Offsetverschiebung in die nächste Zeile von A.
18:	`Sub`	`R13,R13,1`	Anzahl der noch zu bearbeitenden Zeilen.
19:	`GotoGt`	`9,R13`	Wenn diese größer als 0 ist, weitere Zeile bearbeiten.
20:	`Goto`	`29`	Ok, wir haben B in A gefunden.
21:	`Add`	`R7,R7,1`	Neue Position in der Suche in A.
22:	`Sub`	`R9,R9,1`	Anzahl der zu bearbeitenden Spalten.
23:	`GotoGt`	`6,R9`	Wenn noch Spalten zum Bearbeiten übrig sind, dann tue dies.
24:	`Add`	`R7,R7,R3`	Springe (mit Offsetverschiebung in die nächste Zeile) ...
25:	`Sub`	`R7,R7,1`	... Position korrigieren.
26:	`Sub`	`R8,R8,1`	Es wurde eine Zeile bearbeitet.
27:	`GotoGt`	`5,R8`	Sind noch Zeilen übrig ? Wenn ja, dann springen.
28:	`Copy`	`R14,1`	Setze R_{14} auf 1, da nicht gefunden.
29:	`Stop`		Programmende, R_{14} enthält den Wert 0, wenn B in A gefunden wurde.

Lösung zu Aufgabe 38

Für dieses Programm wird erwartet, daß die Parameter folgendermaßen übergeben werden:

- Das Register R_1 enthält den Wert n.
- Im Register R_2 befindet sich die Adresse der Matrix A.
- In R_3 befindet sich die Adresse der Matrix B.
- Das Register R_4 enthält die Adresse berechneten Matrix $A \cdot B$.

Die Größe n der Matrizen sowie die festen Adressen für die Matrix A (1), B $(n^2 + 1)$ und $A \cdot B(2\ n^2 + 1)$ können durch folgende Assembleranweisungen bestimmt werden:

Adr.	Befehl	Op.	Kommentar
1:	`Copy`	`R2,1`	Die Matrix A befindet sich stets an Adresse 1.
2:	`Mul`	`R3,[0],[0]`	Die Adresse von B berechnet sich zu: $n^2 + 1$.
3:	`Add`	`R4,R3,R3`	Die Zielmatrix an der Adresse $2 \cdot n^2 + 1$.
4:	`Add`	`R3,R3,1`	
5:	`Add`	`R4,R4,1`	
6:	`Copy`	`R1,[0]`	Lade das Register R_1 mit dem Wert in Adresse 0.

Um die Darstellung zu vereinfachen, wurde hier die Bezeichnung `[0]` für den Inhalt von Speicherzelle M_0 verwendet. Dies kann offenbar durch die Verwendung eines zusätzlichen Registers umgangen werden. Auch hier wird wieder mit der Technik der Offsetverschiebung gearbeitet. Die Register sind während des Programmablauf folgendermaßen belegt.

- Die aktuelle Position in Matrix A wird im Register R_5 festgehalten.
- Das Register R_6 enthält die aktuelle Position in Matrix B.
- Die Summanden $a_{ik} \cdot b_{kj}$ werden in R_7 aufaddiert, um $(A \cdot B)_{ij}$ zu bestimmen.
- Das Register R_8 enthält die aktuelle Zeilenposition.

Adr.	Befehl	Op.	Kommentar
1:	`Mul`	`R9,R1,R1`	Wir benötigen den Wert $n^2 - 1$.
2:	`Sub`	`R9,R9,1`	Dieser steht im Register R_9.
3:	`Add`	`R5,R2,R9`	Wir beginnen rechts unten in Matrix A.
4:	`Add`	`R6,R3,R9`	Ebenso in der Matrix B...
5:	`Add`	`R4,R4,R9`	.. und Matrix $A \cdot B$.
6:	`Copy`	`R8,R1`	Wir beginnen mit der Zeile n.
7:	`Copy`	`R7,0`	Die Summe $\sum_{k=1}^{n} a_{ik} \cdot b_{kj}$ ist zuerst 0.
8:	`Mul`	`R10,[R5],[R6]`	Bestimme den Wert $a_{ik} \cdot b_{kj}$...
9:	`Add`	`R7,R7,R10`	... und addiere ihn zu R_7.
10:	`Sub`	`R5,R5,R1`	Gehe in die neue Spalte, also n Adressen weiter.
11:	`Sub`	`R6,R6,1`	Gehe in die neue Zeile, also eine Adresse weiter.
12:	`Sub`	`R10,R2,R5`	Wir wollen testen, ob $R_5 \geq R_2$.
13:	`GotoLeq`	`8,R10`	Wenn ja, dann nächste Position berechnen.
14:	`Copy`	`[R4],R7`	Sonst: $(A \cdot B)_{ij}$ eintragen.
15:	`Sub`	`R4,R4,1`	Neue Position in $A \ldots B$.
16:	`Add`	`R5,R5,R9`	Alte Position in B ermitteln.
17:	`Add`	`R6,R6,R1`	Adresse der nächsten Zeile in A berechnen.
18:	`Sub`	`R8,R8,1`	Subtrahiere die Zeilenposition.
19:	`GotoGt`	`7,R8`	Berechne die Spalte zu Ende.
20:	`Sub`	`R6,R6,R1`	Gehe in die nächste Spalte.
21:	`Add`	`R5,R5,R1`	Und in die letzte Zeile von A.
22:	`Sub`	`R10,R6,R3`	Wir wiederholen, solange $R6 > R3$.
23:	`GotoGt`	`6,R10`	Bearbeite ggf. neue Spalte.
24:	`Stop`		Ende

Hierbei wurde zur Vereinfachung der bedingte Sprungbefehl `GotoLeq` benutzt, der ein Register darauf testet, ob der Inhalt *kleiner oder gleich* Null ist.

Lösung zu Aufgabe 39

Bei Verwendung eines *Stacks* kann dieses Problem sehr einfach und elegant gelöst werden. Da die Klammern nie den Wert 0 annehmen können, dient dieser Wert an Adresse $n + 1$ zur Begrenzung des Eingabefeldes und als Endmarke. Der Stack beginnt an der Position $n + 2$ und überschreibt daher die Endmarke nicht. Eine öffnende Klammer wird auf dem Stack abgelegt (beginnend an Position $n + 2$). Wird eine Klammer geschlossen, so wird diese mit dem Klammerwert auf dem Stack verglichen. Ist die Summe der Werte gleich 0, so wurde die Klammer korrekt geschlossen und die öffnende Klammer wird vom Stack entfernt. Anderenfalls ist die Differenz zwischen Ausdruckszeiger R_2 und dem Stackzeiger R_1 ungleich Null, so daß der berechnete Wert für R_3 nicht Null ist. Wird bis zu der Endmarke keine inkorrekte Klammerung festgestellt, so muß noch verifiziert werden, daß alle geöffneten Klammern auch geschlossen wurden. Dies wird dadurch erreicht, daß die Endposition (diese ist $n + 1$) von dem Stackpointer subtrahiert wird. Der Stackpointer ist genau dann gleich $n + 1$, wenn keine Klammer mehr geöffnet ist, so daß R_3 nur dann den Wert Null erhält, wenn die Klammerung vollständig abgeschlossen wurde. Beachte, daß, wenn eine Klammer zuviel geschlossen wird, der Wert dieser Klammer zu 0 (der Endmarke) addiert wird. Dies führt zu einem resultierenden Wert ungleich 0, so daß die inkorrekte Klammerung erkannt wird.

Adr.	Befehl	Op.	Kommentar
1:	`Add`	`R1,[0],1`	Der Stack beginnt an Adresse $n+1$.
2:	`Copy`	`[R1],0`	Die Null dient als Endmarke.
3:	`Copy`	`R2,0`	R_2 enthält die aktuelle Position.
4:	`Add`	`R2,R2,1`	Gehe zur nächsten Position.
5:	`Copy`	`R3,[R2]`	Lade die Klammer an der aktuellen Position.
6:	`GotoEq`	`14,R3`	An der Endmarke angekommen ?
7:	`GotoGt`	`11,R3`	Schließende Klammer ?
8:	`Add`	`R1,R1,1`	Öffnende, also nächste Stackposition.
9:	`Copy`	`[R1],R3`	Trage dort die aktuelle Klammer ein.
10:	`Goto`	`4`	Und fahre mit der nächsten Position fort.
11:	`Add`	`R3,R3,[R1]`	Addiere den Wert der schließenden Klammer auf den der öffnenden auf dem Stack.
12:	`Sub`	`R1,R1,1`	Lösche die öffnende Klammer auf dem Stack.
13:	`GotoEq`	`4,R3`	Korrekte schließende Klammer ?
14:	`Sub`	`R3,R2,R1`	Wenn die aktuelle Position gleich der Stackposition ist, dann ist der Ausdruck korrekt.
15:	`Stop`		R_3 enthält den Wert 0, wenn die Klammerung korrekt war.

Lösung zu Aufgabe 40

Hier ist die Idee einfach: Laufe das Feld beginnend bei 1 durch, sortiere jeweils die Werte $\{a_i, \ldots, a_{i+6}\}$ und trage den viertgrößten Wert in M_i ein. In dem folgenden Programm ist der Wert i stets im Register R_1 gespeichert. Die Werte $\{a_i, \ldots, a_{i+6}\}$ werden in die Speicherzellen $M_{1001}, \ldots M_{1007}$ übertragen und dort mit einem vereinfachten Bubblesortverfahren sortiert. Dieses sucht (absteigend vom Ende des Arrays) nach einem Paar M_j, M_{j+1} mit $M_j > M_{j+1}$ ($j \in \{1001, \ldots 1006\}$), vertauscht dann diese Werte und beginnt erneut beim Ende des Arrays. Wurde kein solches Paar mehr gefunden, so wurde das Array $M_{1001}, \ldots M_{1007}$ aufsteigend sortiert. Der Wert M_{1004} wird dann in den Speicherplatz M_i übertragen und die nächste Stelle bearbeitet.

Adr.	Befehl	Op.	Kommentar
1:	`Copy`	`R1,1`	Wir beginnen mit $i = 1$.
2:	`Copy`	`R4,1000`	Wir benötigen diese Werte zur indirekten...
3:	`Copy`	`R5,1001`	Adressierung im Array.
4:	`Copy`	`R2,7`	Es müssen 7 Werte kopiert werden.
5:	`Sub`	`R2,R2,1`	Subtrahiere die Anzahl um eins.
6:	`Copy`	`[R5+R2],[R1+R2]`	Übertrage den Wert a_{i+j} an die Adresse M_{1001+j}.
7:	`GotoGt`	`5,R2`	Kopiere alle sieben Werte.
8:	`Copy`	`R2,6`	Beginne am Ende des Arrays.
9:	`Sub`	`R3,[R4+R2],[R5+R2]`	Vergleiche die Werte M_j und M_{j+1}.
10:	`GotoLe`	`15,R3`	Wenn der Wert $M_j \leq M_{j+1}$, dann nicht vertauschen.
11:	`Copy`	`R3,[R5+R2]`	Speichere den Wert aus M_{j+1} in R_3.
12:	`Copy`	`[R5+R2],[R4+R2]`	Schreibe den Wert aus M_j nach M_{j+1}.
13:	`Copy`	`[R4+R2],R3`	Schreibe den alten Wert aus M_{j+1} nach M_j.
14:	`Goto`	`8`	Führe eine erneute Arraysuche durch.
15:	`Sub`	`R2,R2,1`	Gehe zur nächsten Arrayadresse.
16:	`GotoGt`	`9,R2`	Und durchsuche das komplette Array.
17:	`Copy`	`[R1],[1004]`	Trage den viertgrößten Wert in a_i ein.
18:	`Add`	`R1,R1,1`	Bearbeite das nächste i.
19:	`Sub`	`R2,995,R1`	Berechne den Wert $995 - i$.
20:	`GotoGt`	`4,R2`	Wiederhole, wenn nötig.
21:	`Stop`		Fertig.

Dieses Assemblerprogramm benötigt sieben zusätzliche Speicherstellen und fünf Register.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

21. April 1998

Blatt 1

Aufgabe 1

Beweisen Sie den folgenden Satz aus der Vorlesung über die b-adische Darstellung natürlicher Zahlen:

Satz: Sei $b \in \mathbb{N}$ mit $b > 0$.
Dann ist jede natürliche Zahl z mit $0 \leq z \leq b^n - 1$ ($n \in \mathbb{N}$) eindeutig als Wort der Länge n über dem Alphabet Σ_b darstellbar durch

$$z = \sum_{i=0}^{n-1} z_i \cdot b^i$$

mit $z_i \in \Sigma_b$ für $i \in \{0, 1, \ldots, n-1\}$.

Hinweis: Hierfür müssen Sie konstruktiv die *Existenz* und mit Hilfe eines Widerspruchsbeweises die *Eindeutigkeit* einer solchen Darstellung nachweisen. **10 Punkte**

Aufgabe 2

Zeigen Sie, daß für die Struktur $(B, +, \cdot, \overline{\ })$ für alle $x, y, z \in B$ die folgenden Gesetze gelten:

(i) $(x + y) \cdot x = x$ (Verschmelzungsgesetze)

$(x \cdot y) + x = x$

(ii) $x + (y \cdot z) = (x + y) \cdot (x + z)$ (Distributivgesetz)

(iii) $\overline{(x + y)} = \overline{x} \cdot \overline{y}$ (de Morgansche Regeln)

$\overline{(x \cdot y)} = \overline{x} + \overline{y}$

(iv) $x = x + x = x \cdot x = \overline{\overline{x}}$

10 Punkte

Aufgabe 3

Zeigen Sie, daß die Multiplikation einer beliebigen natürlichen Zahl mit 8 sowie die Division einer beliebigen natürlichen Zahl durch 8 im Oktalsystem nur eine Stellenverschiebung (Shift) ist. **10 Punkte**

Aufgabe 4

Sei $n = 3m, m \in \mathbb{N}$. Zeigen Sie, daß

$$(z_{n-1}z_{n-2}\ldots z_1z_0)_2 = ((z_{n-1}z_{n-2}z_{n-3})_8 \ldots (z_2z_1z_0)_8)_8$$

gilt, d. h., zeigen Sie, daß

$$\sum_{i=0}^{n-1} z_i \cdot 2^i = \sum_{j=0}^{m-1} (z_{3j+2}z_{3j+1}z_{3j})_8 \cdot 8^j$$

gilt. **10 Punkte**

Aufgabe 5

Stellen Sie die Multiplikation von zwei zweistelligen Dualzahlen a_1a_2 und b_1b_2 als Schaltfunktion $\mathcal{M}_2 : B^4 \to B^4$ dar, d. h. geben Sie für $i \in \{1, \ldots, 4\}$ Boolesche Funktionen $f_i(a_1, a_2, b_1, b_2)$ an, so daß

$$\mathcal{M}_2(a_1, a_2, b_1, b_2) = (f_1(a_1, a_2, b_1, b_2), f_2(a_1, a_2, b_1, b_2), f_3(a_1, a_2, b_1, b_2), f_4(a_1, a_2, b_1, b_2))$$

gilt. **10 Punkte**

Abgabe: Bis **Montag, 27. April 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

28. April 1998

Blatt 2

Aufgabe 6

Zeigen Sie, daß die disjunktive Normalform (DNF) einer Booleschen Funktion in folgendem Sinne eindeutig ist:
Seien $I_1, I_2 \subseteq \{0, \ldots, 2^n - 1\}$. Dann gilt für jede Boolesche Funktion $f : B^n \to B$:

$$f(x_1, \ldots, x_n) = \sum_{i \in I_1} m_i(x_1, \ldots, x_n) = \sum_{i \in I_2} m_i(x_1, \ldots, x_n) \Longrightarrow I_1 = I_2.$$

10 Punkte

Aufgabe 7

Beweisen Sie den folgenden Satz über die Darstellung einer Booleschen Funktion als Konjunktion von Maxtermen:
Satz: Sei $n \in \mathbb{N}, n \geq 1$. Dann läßt sich jede Boolesche Funktion $f : B^n \to B$ darstellen als

$$\begin{aligned} f(x_1, \ldots, x_n) &= \bigwedge_{\substack{(\beta_1, \ldots, \beta_n) \in B^n \\ f(\beta_1, \ldots, \beta_n) = 0}} x_1^{1-\beta_1} + x_2^{1-\beta_2} + \cdots + x_n^{1-\beta_n} \\ &= \prod_{i \in \{0,1,\ldots,2^n-1\} - I_f} M_i(x_1, x_2, \ldots, x_n). \end{aligned}$$

10 Punkte

Aufgabe 8

Zeigen Sie, daß das System $\{\downarrow\}$ funktional vollständig ist. **10 Punkte**

Aufgabe 9

Sei die Boolesche Funktion $f : B^3 \to B$ gegeben durch

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Stellen Sie f dar in

a) disjunktiver Normalform (DNF),

b) konjunktiver Normalform (KNF),

c) Ringsummen-Normalform (RNF),

d) komplementfreier Ringsummendarstellung,

e) einer Darstellung, die nur $\uparrow$ benutzt.

10 Punkte

Aufgabe 10

Sei die Boolesche Funktion $f : B^3 \to B$ gegeben durch

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Konstruieren Sie ein Schaltnetz für f, das nur die Gatter $+, \cdot$ und $\overline{\ }$ benutzt, so daß

a) die Anzahl der Stufen minimal wird,

b) die Anzahl der Gatter minimal wird.

10 Punkte

Abgabe: Bis **Montag, 4. Mai 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

5. Mai 1998

Blatt 3

Aufgabe 11

a) Gegeben seien fertige Bauteile mit der Funktion eines 6-MUX. Konstruieren Sie hieraus einen 8-MUX.

b) Beweisen Sie, daß Ihre Konstruktion korrekt ist.

c) Berechnen Sie die Komplexität (Anzahl der Gatter) Ihres 8-MUX-Entwurfs, ausgehend von der in der Vorlesung beschriebenen Komplexität des 6-MUX.

10 Punkte

Aufgabe 12

Sei $d = 2^{4m}$.

a) Konstruieren Sie rekursiv einen d-MUX aus 2^{3m}-MUX-Bauteilen.

b) Beweisen Sie die Korrektheit Ihrer Konstruktion.

c) Berechnen Sie die Komplexität (Anzahl der Gatter) Ihres d-MUX-Entwurfs.

10 Punkte

Aufgabe 13

Zeigen Sie, daß man für $d \geq 3$ jede d-stellige Boolesche Funktion mit Hilfe eines $(d-1)$-MUX derart realisieren kann, daß das entstehende Schaltnetz höchstens

$$d + 3 \cdot (2^{d-1} - 1)$$

Gatter enthält. **10 Punkte**

Aufgabe 14

Verbessern Sie die Aussage von Satz 2.2, indem Sie beweisen, daß für $d \geq 9$ eine d-stellige Boolesche Funktion f existiert, so daß jedes Schaltnetz für f mindestens

$$c \cdot \frac{2^{d-3}}{d}$$

Knoten für ein $c \in \mathbb{R}, c > 1$ hat.
Versuchen Sie dabei, das c so groß wie möglich zu wählen. **10 Punkte**

Abgabe: Bis **Montag, 11. Mai 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

12. Mai 1998

Blatt 4

Aufgabe 15

Konstruieren Sie für $d \in \mathbb{N}$ einen $d \times 2^d$-Decoder, der Gatter von beliebigem Fan-in benutzen darf. **10 Punkte**

Aufgabe 16

a) Berechnen Sie die Anzahl der benötigten Gatter und die Zeitkomplexität eines 8-stelligen Carry-Select-Addiernetzes, das nur Gatter mit Fan-in ≤ 2 benutzt.

b) Entwerfen Sie ein 12-stelliges Carry-Select-Addiernetz mit möglichst geringer Zeitkomplexität. Hierbei dürfen Sie Gatter mit beliebigem Fan-in benutzen.

10 Punkte

Aufgabe 17

Konstruieren Sie ein Schaltnetz für die Addition von 9 n-stelligen Dualzahlen mit möglichst geringer Zeitkomplexität. **10 Punkte**

Aufgabe 18

a) Entwerfen Sie ein Schaltnetz für einen Halbaddierer (Resultat und Übertrag), der ausschließlich aus NOR-Gattern mit 2 Eingängen besteht und begründen Sie die Korrektheit Ihres Entwurfs. Verwenden Sie nicht mehr als 5 NOR-Gatter.

b) Konstruieren Sie einen ausschließlich aus NOR-Gattern bestehenden Volladdierer (unter Verwendung der in a) konstruierten Halbaddierer).

c) Wir nehmen nun an, daß ein NOR-Gatter eine Schaltzeit von 20 psec hat. Wann liegt bei Ihrem Halbaddierer das Resultat vor? Wann der Übertrag? Wie sehen die Zeiten für Ihren Volladdierer aus?

10 Punkte

Abgabe: Bis **Montag, 18. Mai 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

19. Mai 1998

Blatt 5

Aufgabe 19

Gegeben sei die folgende Boolesche Funktion f:

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$f(x_1, \ldots, x_4)$	1	1	0	1	0	0	0	0	1	1	0	1	1	1	0	1

a) Wenden Sie das Karnaugh-Verfahren auf die DNF von f an, um eine möglichst kurze disjunktive Form von f zu erhalten.

b) Zeigen Sie, daß die in a) gefundene Darstellung $\mathrm{DF}_{\min}$ in folgendem Sinne nicht optimal ist:
Es existiert eine Boolesche Formel mit $< \mathrm{Kost}(\mathrm{DF}_{\min})$ zweistelligen Operatoren, die f beschreibt.

10 Punkte

Aufgabe 20

Gegeben sei die folgende Boolesche Funktion f:

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$f(x_1, \ldots, x_4)$	0	0	1	1	0	1	0	1	1	0	1	0	1	1	0	0

a) Bestimmen Sie alle Implikanten von f.

b) Bestimmen Sie alle Primimplikanten von f.

c) Bestimmen Sie zwei unterschiedliche disjunktive Formen (DF) von f, die beide das Ergebnis des Quine-McCluskey-Verfahrens sein können.

10 Punkte

Aufgabe 21

a) Entwerfen Sie ein Schaltwerk für die Addition von sechs 8-stelligen Dualzahlen, ausgehend von einem 11-stelligen Carry-Select-Addier-Schaltnetz.

b) Beschreiben Sie Ihr Schaltwerk aus a) durch einen Mealy-Automaten.

10 Punkte

Aufgabe 22

In der Vorlesung wurde ein linearer Schaltkreis für die Polynommultiplikation angegeben, der ein beliebiges Polynom $a(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ mit einem *fest* vorgegebenen Polynom $h(x) = h_0 + h_1 x + \cdots + h_{n-k} x^{n-k}$ multipliziert.
Beschreiben Sie dieses Schaltwerk durch einen Mealy-Automaten. **10 Punkte**

Abgabe: Bis **Montag, 25. Mai 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

26. Mai 1998

Blatt 6

Aufgabe 23

Gegeben sei das folgende Problem, im weiteren kurz WZK-Problem genannt:

> Ein Farmer steht mit einem Wolf, einer Ziege und einem Kohlkopf am Ufer eines Flusses und möchte diesen Fluß überqueren. Dafür steht ihm aber nur ein kleines Boot zur Verfügung, das nur den Farmer und einen weiteren Passagier bzw. Gegenstand tragen kann. Weiterhin darf der Farmer nicht den Wolf und die Ziege oder die Ziege und den Kohlkopf unbeaufsichtigt an demselben Ufer zurücklassen, da sonst der Wolf die Ziege oder die Ziege den Kohlkopf frißt.

a) Beschreiben Sie formal einen Mealy-Automaten, der das WZK-Problem löst. Sie können hierfür z. B. 6 Register mit der folgenden Bedeutung verwenden: Je ein Register beschreibt, ob sich der Wolf, die Ziege, der Kohlkopf oder das Boot am linken Ufer befinden und die restlichen zwei Register beschreiben den Inhalt des Bootes. Das Ausgabealphabet könnte in diesem Fall aus den folgenden 5 Buchstaben bestehen: $\Delta = \{$ L_W (Lade den Wolf in das Boot),
L_Z (Lade die Ziege in das Boot),
L_K (Lade den Kohlkopf in das Boot),
E (Entlade das Boot),
F (Fahre das Boot über den Fluß)$\}$

b) Entwerfen Sie ein Schaltwerk für das WZK-Problem.

10 Punkte

Aufgabe 24

Seien $x, y \in \{0, 1, \ldots, 2^n - 1\}$.
Beweisen Sie, daß $K_2((K_2(x))_2 + (K_2(y))_2 \bmod 2^n)$ die Zweier-Komplement-Darstellung von $-x - y$ ist.

10 Punkte

Aufgabe 25

Wie groß ist die größte Zahl, die in Gleitkomma-Darstellung mit Basis $b = 2$, Mantissenlänge 23 und Exponentenlänge 8 darstellbar ist? Wie groß ist die absolut kleinste Zahl (d. h. die kleinste positive Zahl), die in dieser Gleitkomma-Darstellung darstellbar ist? Wie groß kann der Rundungsfehler für die Zahlen aus diesem Intervall sein?
Vergleichen Sie diese Ergebnisse mit der Festkomma-Darstellung mit $b = 2$, 23 Vorkomma-Bits und 8 Nachkomma-Bits. **10 Punkte**

Aufgabe 26

Beschreiben Sie das Multiplikationswerk zur Multiplikation von zwei Dualzahlen der Länge n aus der Vorlesung durch einen Mealy-Automaten. **10 Punkte**

Aufgabe 27

a) Konstruieren Sie ein Schaltwerk für die Multiplikation von 7-stelligen ganzen positiven Dualzahlen, das dadurch eine Beschleunigung gegenüber dem Multiplikationswerk aus der Vorlesung erreicht, daß ein Carry-Save-Addiernetz verwendet wird.

b) Beschreiben Sie Ihr Schaltwerk aus a) durch einen Mealy-Automaten.

10 Punkte

Aufgabe 28

Entwerfen Sie ein Addierwerk für die Addition von zwei ganzen Zahlen in Zweier-Komplement-Darstellung. Dabei soll auch das Resultat in Zweier-Komplement-Darstellung ausgegeben werden. **10 Punkte**

Abgabe: Bis **Montag, 8. Juni 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

9. Juni 1998

Blatt 7

Aufgabe 29

a) Schreiben Sie ein Programm in der in der Vorlesung vorgestellten Assemblersprache, das drei in den Registern R_1, R_2 und R_3 stehende Zahlen aufsteigend sortiert.

b) Gegeben seien 100 aufsteigend geordnete Zahlen in den Speicherzellen $M_1, \ldots, M_{100}$. Schreiben Sie ein Assemblerprogramm, das diese Zahlen absteigend ordnet. Verwenden Sie dabei maximal drei zusätzliche Register.

10 Punkte

Aufgabe 30

Gegeben seien 100 Zahlen in den Speicherzellen $M_1, \ldots, M_{100}$. Schreiben Sie ein Assemblerprogramm, das das Minimum und das Maximum dieser Zahlen berechnet. Das Minimum (bzw. Maximum) soll am Ende der Rechnung in der Speicherzelle M_{101} (bzw. M_{102}) stehen. Verwenden Sie dabei maximal drei zusätzliche Register. **10 Punkte**

Aufgabe 31

Die *Fibonacci-Zahlen* $F(i)$ sind für $i \in \mathbb{N}$ folgendermaßen rekursiv definiert:

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(i) &= F(i-1) + F(i-2) \quad \text{für } n \geq 3. \end{aligned}$$

Schreiben Sie ein Assemblerprogramm, das die n-te Fibonacci-Zahl berechnet, wobei n durch den Inhalt des Registers R_0 gegeben sei. Ihr Programm sollte dabei für beliebig große Werte von n nur höchstens vier Register verwenden. **10 Punkte**

Aufgabe 32

Bei einem Von-Neumann-Rechner liegen Programm und Daten in demselben Speicher. Bisher wurde in der Vorlesung davon ausgegangen, daß das Programm nur auf den Daten operiert. Es ist aber auch denkbar, daß das Programm sich selbst modifiziert (vgl. z. B. Computerviren).

Beweisen oder widerlegen Sie:

Ein Von-Neumann-Rechner mit endlichem Speicher und selbstmodifizierendem Programm kann durch einen Mealy-Automaten beschrieben werden. **10 Punkte**

Abgabe: Bis **Montag, 15. Juni 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

16. Juni 1998

Blatt 8

Aufgabe 33

Das folgende Assemblerprogramm soll alle Primzahlen berechnen, die kleiner oder gleich der in der Speicherzelle M_0 gegebenen natürlichen Zahl $n \geq 2$ sind. Als Ergebnis soll in der Speicherzelle M_i eine 1 stehen, wenn i prim ist und sonst eine 0 für alle $1 \leq i \leq n$.

```
 1: LoadM 0      8: Load 2      15: StoreM 1    22: StoreR 0
 2: StoreR 0     9: StoreR 0    16: StoreIR 1   23: SubM 0
 3: Load 1      10: LoadM 0     17: LoadR 1     24: GotoGt 28
 4: StoreIR 0   11: DivR 0      18: SubR 0      25: LoadIR 0
 5: LoadR 0     12: MulR 0      19: GotoGt 13   26: GotoGt 9
 6: Sub 1       13: StoreR 1    20: LoadR 0     27: Goto 20
 7: GotoGt 2    14: Load 0      21: Add 1       28: Stop
```

a) Beschreiben Sie die Situation (Speicher und Register) nach der ersten Ausführung von Zeile 9.

b) Leider ist dem Programmierer ein Fehler unterlaufen. Finden Sie diesen Fehler und korrigieren Sie das Programm. (Hinweis: Es reicht aus, zwei zusätzliche Zeilen einzufügen und die Sprungadressen geeignet an die neue Numerierung anzupassen.)

c) Kommentieren Sie das korrekte Programm.

Bemerkung zu der Idee des Programms:
Das Programm benutzt das *Sieb des Erathostenes*, d. h. es basiert auf folgender Beobachtung: Falls p eine Primzahl ist, so ist jede Zahl der Form $i \cdot p$ für $i > 1$ keine Primzahl.

10 Punkte

Aufgabe 34

Gegeben seien 100 Zahlen $a_1, \ldots, a_{100}$ in den Speicherzellen $M_1, \ldots, M_{100}$. Schreiben Sie ein Programm in dem in der Vorlesung vorgestellten erweiterten Assembler, das die Teilsummen $s_i = \sum_{j=\max\{1,i-9\}}^{i} a_j$ für $i \in \{1, \ldots, 100\}$ berechnet und s_i in die Speicherzelle M_{100+i} schreibt.
Die Ausführung Ihres Programms sollte deutlich weniger als 900 Additionen benötigen.

10 Punkte

Aufgabe 35

Geben Sie für folgende Operationen ein Mikroprogramm anhand des in der Vorlesung vorgestellten (und als Kopie verteilten) CPU-Modells an.

a) $\delta(\alpha) := \delta(M_{\delta(R_0)})$,

b) $\delta(\alpha) := \max\{\delta(R_0), \delta(R_1)\}$,

c) $\delta(\alpha) := \delta(M_{\delta(R_0)\cdot 4+\delta(R_1)})$.

10 Punkte

Aufgabe 36

In den Speicherzellen M_i stehe jeweils entweder eine 1 oder eine 0 für alle i.
Schreiben Sie ein Programm in dem in der Vorlesung vorgestellten erweiterten Assembler, das das kleinste i bestimmt, für das es ein $k \geq 1$ gibt mit den folgenden Eigenschaften:

- $\delta(M_i) = 1$ und
- $\delta(M_j) = 0$ für alle $i + 1 \leq j \leq i + k$ und
- $\delta(M_j) = 1$ für alle $i + k + 1 \leq j \leq i + 2k$.

10 Punkte

Abgabe: Bis **Montag, 22. Juni 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.

RWTH AACHEN
Lehrstuhl für Informatik I
Prof. J. Hromkovič

Ahornstr. 55
52056 Aachen

Übungen zur Vorlesung „Rechnerstrukturen“

23. Juni 1998

Blatt 9

Aufgabe 37

Im Speicher seien eine $k \times \ell$-Matrix $A = (a_{ij})$ und eine $k' \times \ell'$-Matrix $B = (b_{ij})$ mit $1 \leq k' \leq k$ und $1 \leq \ell' \leq \ell$ folgendermaßen gegeben:
$k = \delta(M_0), \quad \ell = \delta(M_1), \quad k' = \delta(M_2), \quad \ell' = \delta(M_3),$
$a_{ij} = \delta(M_{i+(j-1)\cdot k+3}), \quad b_{ij} = \delta(M_{k\cdot\ell+i+(j-1)\cdot k'+3}).$
Schreiben Sie ein Programm in dem erweiterten orthogonalen Assembler der Vorlesung, das testet, ob B eine Teilmatrix von A ist. Kommentieren Sie Ihr Programm. **10 Punkte**

Aufgabe 38

Im Speicher seien zwei $n \times n$-Matrizen A und B folgendermaßen gegeben:
$n = \delta(M_0), \quad a_{ij} = \delta(M_{i+(j-1)\cdot n}), \quad b_{ij} = \delta(M_{n^2+i+(j-1)\cdot n}).$
Schreiben Sie ein Programm in dem erweiterten orthogonalen Assembler aus der Vorlesung, das die Produktmatrix $A \cdot B = C = (c_{ij})$ berechnet und wie folgt im Speicher ablegt:
$c_{ij} = \delta(M_{2\cdot n^2+i+(j-1)\cdot n})$.
Kommentieren Sie Ihr Programm. **10 Punkte**

Aufgabe 39

Gegeben sei eine Menge von Klammersymbolen $A = \{(_1, (_2, \ldots, (_k,)_1,)_2, \ldots,)_k\}$. Die Menge der *korrekt geklammerten Ausdrücke über A* ist rekursiv definiert durch:

1. $(_i\ )_i$ ist ein korrekt geklammerter Ausdruck für alle $1 \leq i \leq k$.
2. Falls a und b korrekt geklammerte Ausdrücke sind, dann sind auch ab und $(_i\ a\ )_i$ korrekt geklammerte Ausdrücke für alle $1 \leq i \leq k$.

In M_0 sei eine natürliche Zahl n gegeben und in $M_1, \ldots, M_n$ stehe jeweils ein Klammersymbol aus A. (Dabei sei $(_i$ kodiert durch $-i$ und $)_i$ durch $+i$ für alle $1 \leq i \leq k$.)
Schreiben Sie ein Programm in dem erweiterten orthogonalen Assembler der Vorlesung, das entscheidet, ob $\delta(M_1) \ldots \delta(M_n)$ ein korrekt geklammerter Ausdruck ist. Beschreiben Sie die Funktionsweise Ihres Programms und kommentieren Sie es. **10 Punkte**

Aufgabe 40

In den Speicherzellen $M_1, \ldots, M_{1000}$ seien die natürlichen Zahlen $a_1, \ldots, a_{1000}$ gegeben. Schreiben Sie ein Programm in dem erweiterten orthogonalen Assembler der Vorlesung, das in M_i den viertgrößten Wert aus $\{a_i, \ldots, a_{i+6}\}$ schreibt für alle $i \in \{1, \ldots, 994\}$.
Ihr Programm sollte nur sieben zusätzliche Speicherzellen und höchstens zehn Register verwenden. Beschreiben Sie die Funktionsweise Ihres Programms und kommentieren Sie es. **10 Punkte**

Abgabe: Bis **Montag, 29. Juni 1998, 12.00 Uhr** in den Sammelkasten vor dem Sekretariat des Lehrstuhls.