technische universität
dortmund

fakultät für informatik

Rechnerstrukturen, Teil 1

Vorlesung 4 SWS WS 16/17

Prof. Dr. Jian-Jia Chen
Fakultät für Informatik – Technische Universität Dortmund
jian-jia.chen@cs.uni-dortmund.de
http://ls12-www.cs.tu-dortmund.de

Übersicht

1. Organisatorisches ✓
2. Einleitung ✓
3. Repräsentation von Daten ✓
4. Boolesche Funktionen und Schaltnetze ✓
5. **Rechnerarithmetik**
6. Optimierung von Schaltnetzen
7. Programmierbare Bausteine
8. Synchrone Schaltwerke

5. Rechnerarithmetik

1. Einleitung
2. Addition natürlicher Zahlen
3. Multiplikation natürlicher Zahlen
4. Addition ganzer Zahlen
5. Addition von Fließkommazahlen
6. Multiplikation von Fließkommazahlen

5.1 Einleitung

Rechnen zu können ist für Computer zentral.

Grundlegende Rechenoperationen sollten durch die Hardware unterstützt werden.

Betrachtete Rechenoperationen

- Addition
- Subtraktion
- Multiplikation
- keine Division

Betrachtete Datentypen

- mit natürlichen Zahlen
- mit ganzen Zahlen
- mit rationalen Zahlen (IEEE 754-1985)

5. Rechnerarithmetik

1. Einleitung ✓
2. **Addition natürlicher Zahlen**
3. Multiplikation natürlicher Zahlen
4. Addition ganzer Zahlen
5. Addition von Fließkommazahlen
6. Multiplikation von Fließkommazahlen

5.2 Addition natürlicher Zahlen

Schulalgorithmus für die Addition von Dezimalzahlen

1. Summand		9	3	8	9	9	8	9
2. Summand			9	7	9	8	9	8
Übertrag	1	1	1	1	1	1	1	
Summe	1	0	3	6	9	8	8	7

Übertragung auf Binärzahlen

1. Summand		1	1	0	0	1	1	1
2. Summand		1	0	0	1	0	1	1
Übertrag	1			1	1	1	1	
Summe	1	0	1	1	0	0	1	0

5.2 Addition natürlicher Zahlen

Addition von Binärzahlen

Beobachtung zu den Überträgen

- 1 + 1 erzeugt einen Übertrag
- 0 + 0 eliminiert einen vorhandenen Übertrag
- 0 + 1 und 1 + 0 reichen einen vorhandenen Übertrag weiter
- Übertrag ist höchstens 1

Kann man Addition als boolesche Funktion ausdrücken?

- **Summenbit** *s*
- **Übertrag** *c* (engl. *carry*)

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $f_{HA}: B^2 \rightarrow B^2$
- f_{HA} realisiert die Forderungen zum Teil

5.2 Addition natürlicher Zahlen

Addition von Binärzahlen

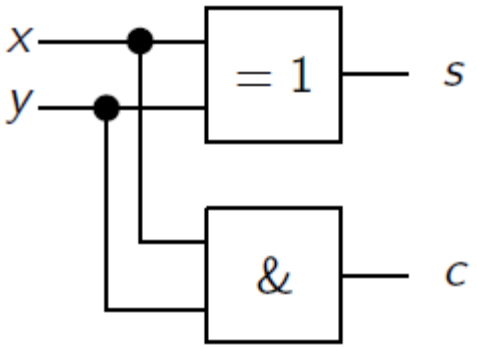
f_{HA} realisiert einen sog. Halbaddierer

Schaltung für den Halbaddierer

$f_{HA}: B^2 \rightarrow B^2$

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$c = x \wedge y \quad s = x \oplus y$



Beobachtung

- gültig nur für isolierte Ziffern
- Berücksichtigung des vorherigen Übertrags fehlt

5.2 Addition natürlicher Zahlen

Halbaddierer

- berechnet Übertrag aus der Addition
- berücksichtigt keinen Übertrag, der schon vorher entstanden ist

Volladdierer

- $f_{VA}: B^3 \rightarrow B^2$
- berechnet Übertrag c aus der Addition
- berücksichtigt Übertrag c_{alt}, der schon vorher entstanden ist

c_{alt}	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

5.2 Addition natürlicher Zahlen

Volladdierer

c_{alt}	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Beobachtung für s

- $s = 1 \Leftrightarrow$ Anzahl der Einsen ungerade
- $s = c_{alt} \oplus x \oplus y$

Beobachtung für c

- $c = 1 \Leftrightarrow$ Anzahl Einsen ≥ 2
- DNF: $c = \overline{c}_{alt} xy \vee c_{alt} \overline{x} y \vee c_{alt}\, x\overline{y} \vee c_{alt} xy$
- Resolution
  - $\overline{c}_{alt} xy \vee c_{alt} xy = xy$
  - $c_{alt} xy \vee c_{alt} \overline{x} y = c_{alt} y$
  - $c_{alt} xy \vee c_{alt} x\overline{y} = c_{alt} x$
- $\boldsymbol{c = xy \vee c_{alt} y \vee c_{alt}\, x}$

5.2 Addition natürlicher Zahlen

Volladdierer

- verbesserte Realisierung für c
- durch Wiederverwendung von Teilergebnissen

c_{alt}	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Es gilt bisher $c = xy \vee c_{alt}y \vee c_{alt}\,x$

Wir müssen die Belegungen für
$(c_{alt}, x, y) \in \{(011), (101), (110), (111)\}$ realisieren

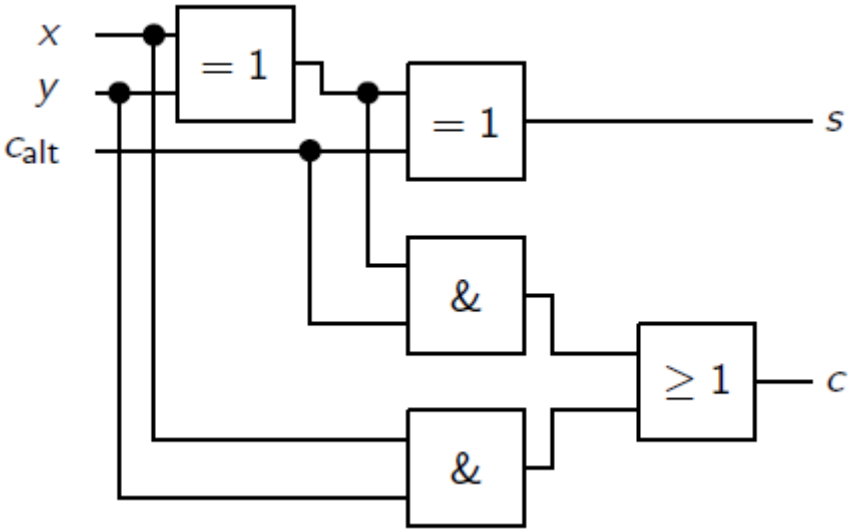
- $x \oplus y = 1 \Leftrightarrow$ genau eine 1 in x, y
- also $c_{alt}(x \oplus y)$ realisiert c für $\{(101), (110)\}$
- fehlen noch die Belegungen $\{(011), (111)\}$ realisiert durch xy

$\rightarrow c = c_{alt}(x \oplus y) \vee xy$

5.2 Addition natürlicher Zahlen

Schaltnetz Volladdierer

- $s = c_{alt} \oplus x \oplus y$
- $c = c_{alt}(x \oplus y) \vee xy$
- Größe 5
- Tiefe 3

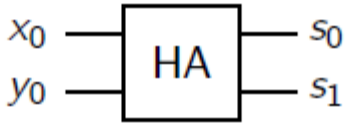


c_{alt}	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

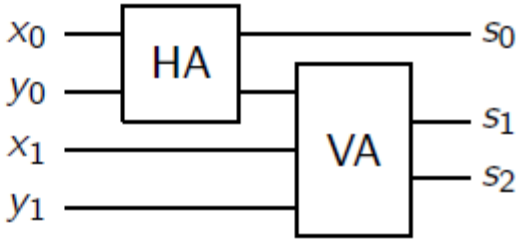
5.2 Addition natürlicher Zahlen

Vollständiger Addierer

$$\begin{array}{rcc} & x_0 \\ + & y_0 \\ \hline s_1 & s_0 \end{array}$$



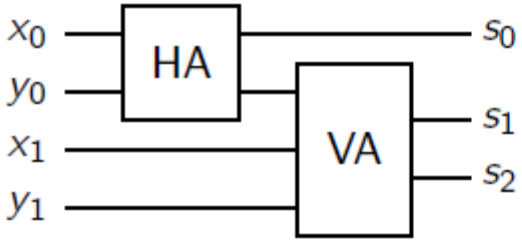
$$\begin{array}{rccc} & x_1 & x_0 \\ + & y_1 & y_0 \\ \hline s_2 & s_1 & s_0 \end{array}$$



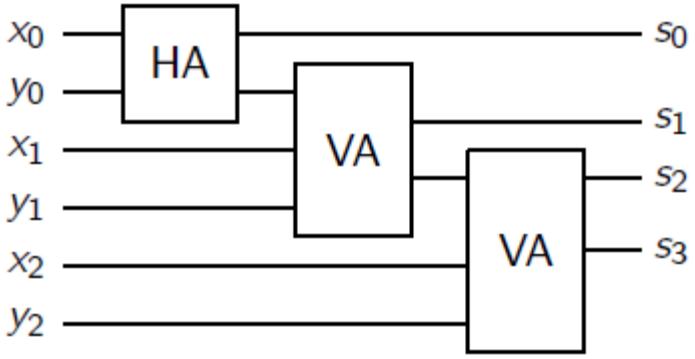
5.2 Addition natürlicher Zahlen

Vollständiger Addierer

$$
\begin{array}{cccc}
 & & x_1 & x_0 \\
+ & & y_1 & y_0 \\
\hline
 & s_2 & s_1 & s_0
\end{array}
$$

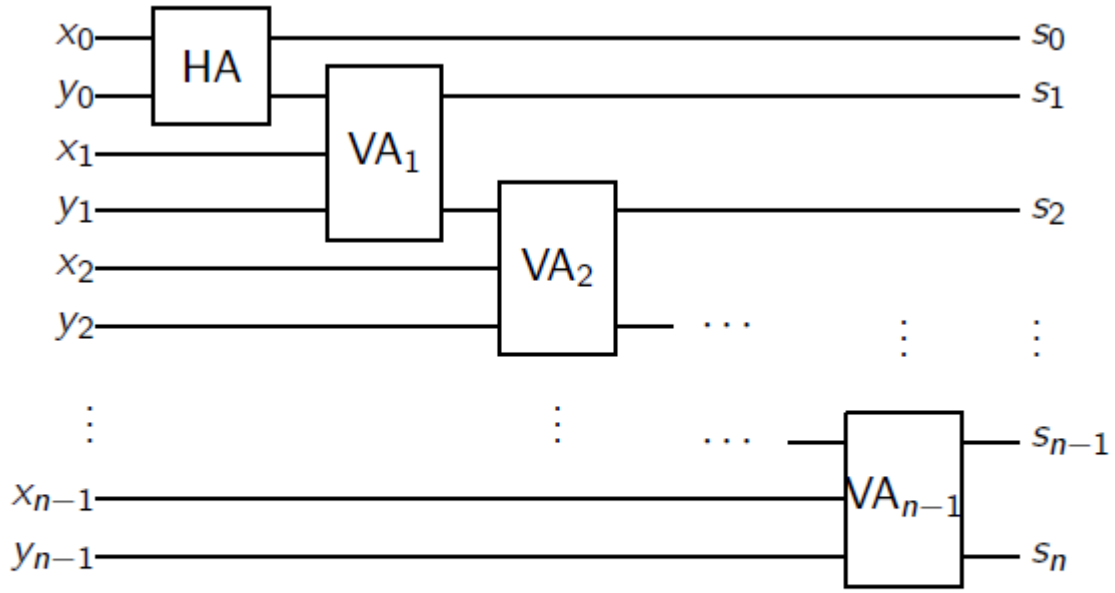


$$
\begin{array}{ccccc}
 & & x_2 & x_1 & x_0 \\
+ & & y_2 & y_1 & y_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$



5.2 Addition natürlicher Zahlen

Vollständiger Addierer



Größe $2 + (n-1) \cdot 5 = 5n - 3$

Tiefe $1 + (n-1) \cdot 3 = 3n - 2$

5.2 Addition natürlicher Zahlen

Ergebnis: Ripple-Carry Addierer

Realisierung „Addition von natürlichen Zahlen“

- Sehr gut strukturiert
- Größe $5n - 3$ **sehr klein**
- Tiefe $3n - 2$ **viel** zu tief

Warum ist unser Schaltnetz so tief?

- Offensichtlich gilt: Überträge brauchen sehr lange
- Verbesserungsidee: **Überträge früh berechnen**
- Wie? Struktureinsicht

5.2 Addition natürlicher Zahlen

Ausnutzen von Einsichten

Struktureinsicht

- $(x_i, y_i) = (1, 1)$ **generiert** Übertrag
- $(x_i, y_i) = (0, 0)$ **eliminiert** Übertrag
- $(x_i, y_i) \in \{(0, 1), (1, 0)\}$ **reicht** Übertrag **weiter**

Dieses Verhalten ist sehr gut durch einen **Halbaddierer** realisierbar

- $u_i = 1$ **generiert** Übertrag
- $v_i = 1$ **reicht** Übertrag **weiter**

x_i	y_i	u_i	v_i	Übertrag
0	0	0	0	eliminieren
0	1	0	1	weiterreichen
1	0	0	1	weiterreichen
1	1	1	0	generieren

zentrale Beobachtung:

- jeder HA in **Tiefe 1** parallel realisierbar berechnet u_i und v_i

5.2 Addition natürlicher Zahlen

Carry Look-Ahead Addierer

Notation

- $f_{HA}: B^2 \rightarrow B^2,\ f_{HA}(x_i, y_i) = (u_i, v_i)$
- u_i = 1 **generiert** Übertrag, v_i = 1 **reicht** Übertrag **weiter**

Vorausberechnung der Überträge

$$c_i = u_{i-1} \vee c_{i-1} v_{i-1}$$

5.2 Addition natürlicher Zahlen

Carry Look-Ahead Addierer

Notation

- $f_{HA}: B^2 \to B^2,\ f_{HA}(x_i, y_i) = (u_i, v_i)$
- u_i = 1 **generiert** Übertrag, v_i = 1 **reicht** Übertrag **weiter**

Vorausberechnung der Überträge

$$c_1 = u_0$$

$$c_2 = u_1 \vee u_0 v_1$$

$$c_3 = u_2 \vee u_1 v_2 \vee u_0 v_1 v_2$$

$$c_4 = u_3 \vee u_2 v_3 \vee u_1 v_2 v_3 \vee u_0 v_1 v_2 v_3$$

$$c_i = u_{i-1} \vee u_{i-2} v_{i-1} \vee u_{i-3} v_{i-2} v_{i-1} \vee \ldots \vee u_0 v_1 v_2 \ldots v_{i-1}$$

$$c_i = \bigvee_{j=0}^{i-1} \left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)$$

5.2 Addition natürlicher Zahlen

Carry Look-Ahead Addierer

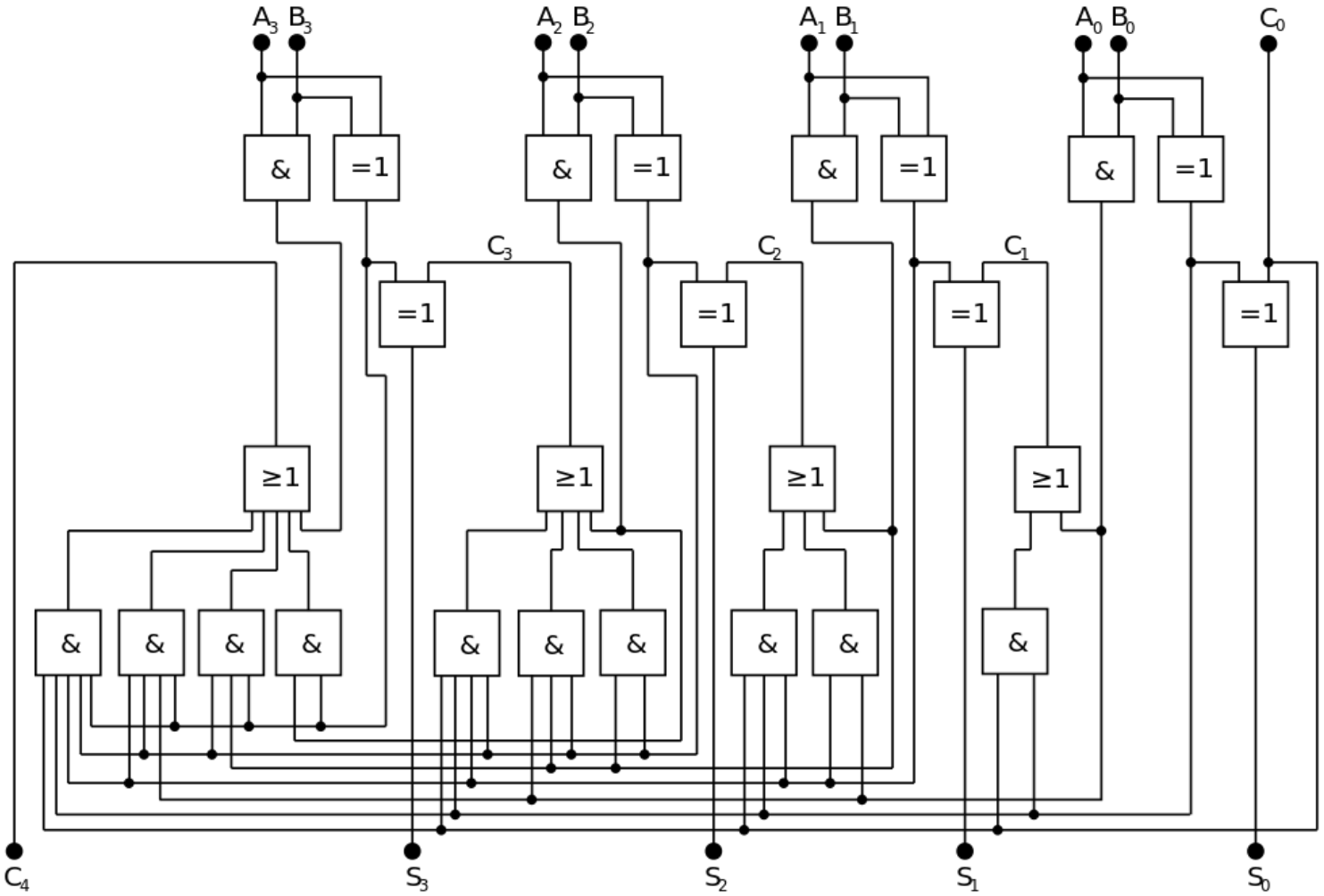
$$c_i = \bigvee_{j=0}^{i-1} \left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)$$

Gesamttiefe des CLA: 4

- alle Halbaddierer (u_i, v_i) Tiefe 1
- alle Und-Gatter $\left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)$ Tiefe 1
- großes Oder-Gatter Tiefe 1
- n× $\oplus$-Gatter für korr. Summenbits Tiefe 1

5.2 Addition natürlicher Zahlen

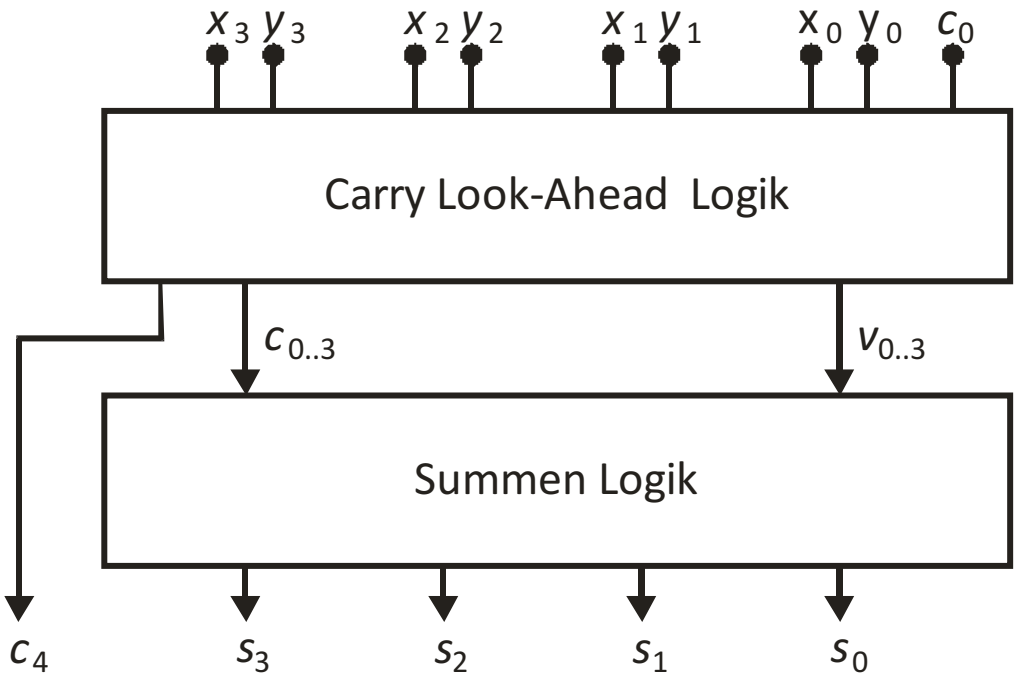
Carry Look-Ahead Addierer



Quelle: Denis Pitzschel

5.2 Addition natürlicher Zahlen

Carry Look-Ahead Addierer



Quelle: Denis Pitzschel

5.2 Addition natürlicher Zahlen

Carry Look Ahead Addierer

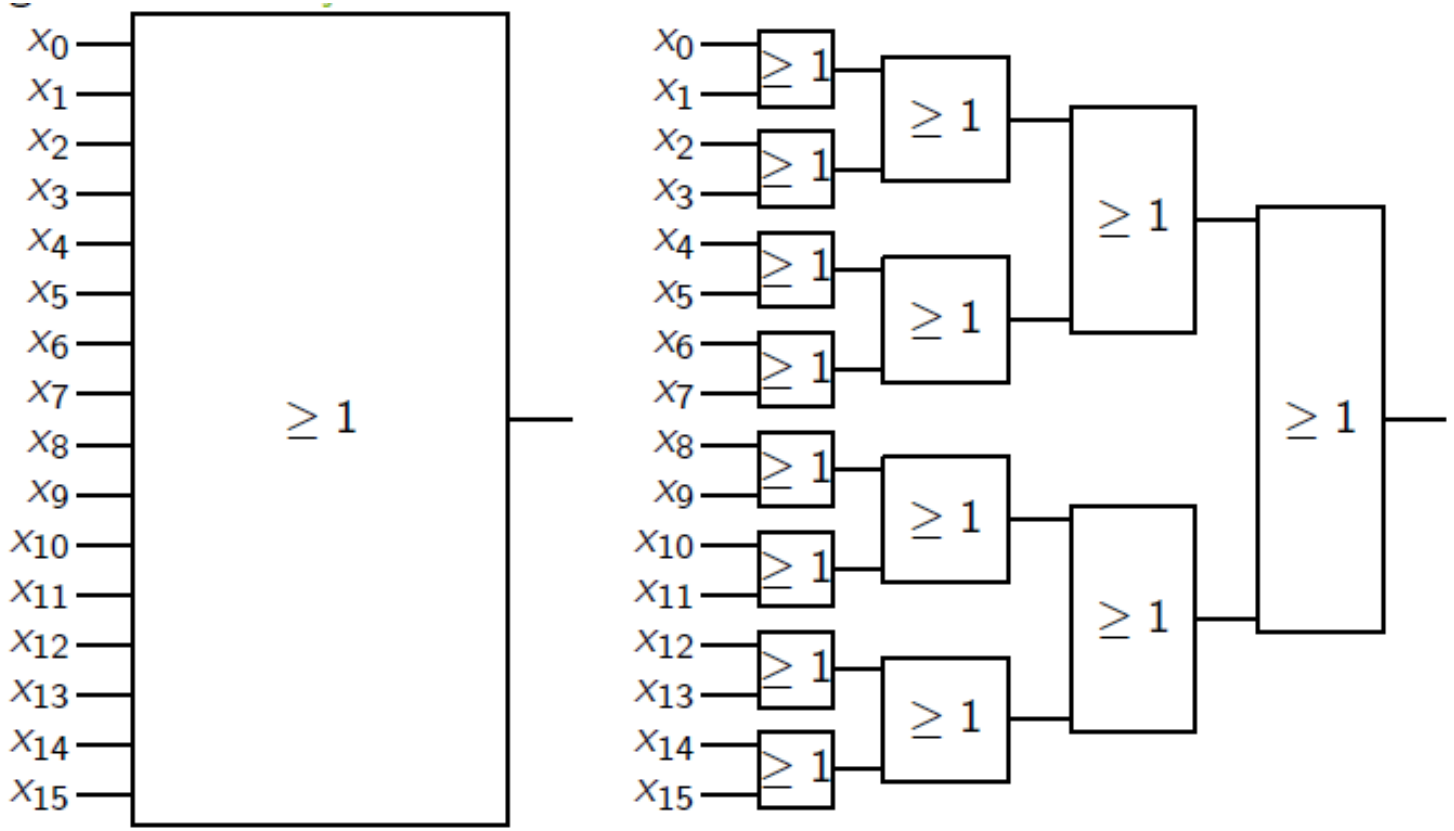
Waren wir wirklich fair bei der Bestimmung der Ebenen?

- Beliebig lange Zahlen in Tiefe 4 addieren. . .
- Anzahl Eingänge eines Gatters heißt **Fan-In**
- Wir vergleichen Schaltnetz mit
  - max. Fan-In 2 (**Ripple-Carry Addierer**)
  - mit Schaltnetz mit max. Fan-In *n* (**Carry Look-Ahead Addierer**)
- Große Fan-Ins sind technologisch schwierig zu realisieren
- **Ziel** Vergleichbarkeit herstellen

5.2 Addition natürlicher Zahlen

Vermeidung von großem Fan-In

Großen Fan-In systematisch verkleinern



Beispiel ODER, Fan-In 16

5.2 Addition natürlicher Zahlen

Vermeidung von großem Fan-In

am Beispiel

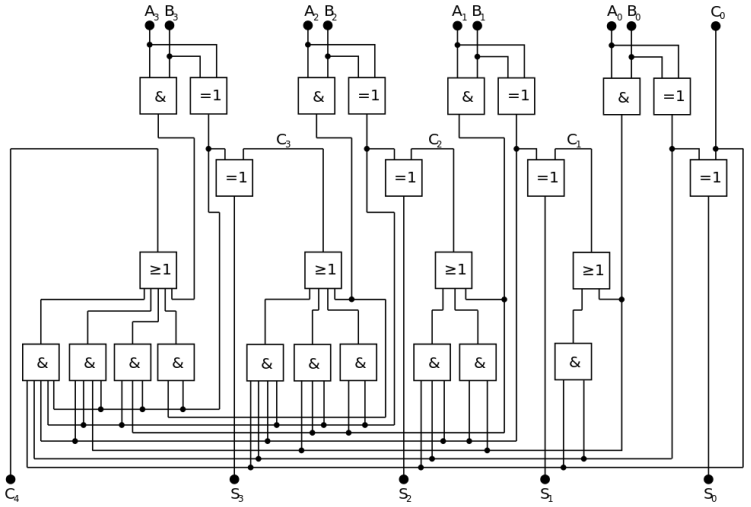
- aus Fan-In 16 bei Tiefe 1 (Größe 1)
- wird Fan-In 2 bei Tiefe 4 (Größe 15)

Wie ist das allgemein?

- **Größe**
  - bei jeder Stufe halb so viele Gatter wie in der Stufe davor
  - $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1 \approx n$ *(geometrische Reihe)*
- **Tiefe**
  - so oft halbieren, bis man bei 1 Gatter ist
  - $\approx \log_2 n$

5.2 Addition natürlicher Zahlen

Carry Look-Ahead Addierer (CLA)



Realisierung der Addition von natürlichen Zahlen

- gut strukturiert
- Größe ≈ n^2 ← ziemlich groß
- Tiefe ≈ $2 \log_2(n)$ ← ziemlich flach

Vergleich zum Ripple Carry Addierer

n	Ripple-Carry Größe	Ripple-Carry Tiefe	Carry Look-Ahead Größe	Carry Look-Ahead Tiefe
8	37	22	64	6
16	77	46	256	8
32	157	94	1 024	10
64	317	190	4 096	12

5.2 Addition natürlicher Zahlen

Einschub – Wie entsteht eine digitale Schaltung?

Video der Firma ELMOS, Dortmund

http://www.youtube.com/watch?v=kuANgMCRnqY

5. Rechnerarithmetik

1. Einleitung ✓
2. Addition natürlicher Zahlen ✓
3. **Multiplikation natürlicher Zahlen**
4. Addition ganzer Zahlen
5. Addition von Fließkommazahlen
6. Multiplikation von Fließkommazahlen

5.3 Multiplikation

Multiplikation

direkt mit Binärzahlen. . .

1	0	1	0	1	1	0	0	·	1	1	1	0	1	0
														0
						1	0	1	0	1	1	0	0	
												0		
				1	0	1	0	1	1	0	0			
			1	0	1	0	1	1	0	0				
		1	0	1	0	1	1	0	0					
	1	0	0	1	1	0	1	1	1	1	1	0	0	0

Multiplizieren heißt

- Nullen passend schreiben
- Zahlen passend verschoben kopieren
- viele Zahlen addieren

5.3 Multiplikation

Multiplikation als Schaltnetz

Multiplikation ist

- Nullen passend schreiben
- Zahlen passend verschoben kopieren
- viele Zahlen addieren

Beobachtung

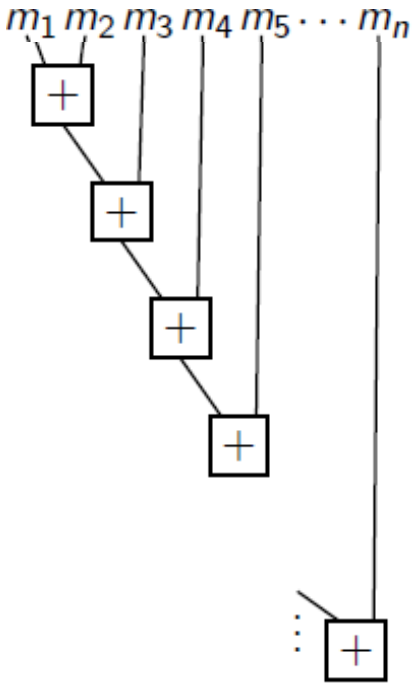
- Nullen schreiben **einfach** und **kostenlos**,
- Zahlen verschieben und kopieren **einfach** und **kostenlos**,
- **viele** Zahlen addieren nicht ganz so einfach

5.3 Multiplikation

Addition vieler Zahlen

- Wir haben Addierer für die Addition zweier Zahlen.
- Wie addieren wir damit *n* Zahlen?

Erster Ansatz : einfach nacheinander



Bewertung

- Schema hat die Tiefe (n − 1)
- in jeder Ebene noch die Tiefe des Addierwerks
- Tiefe = (n-1) · Tiefe(Addierer)
- Tiefe (und damit die Laufzeit) ist zu groß!

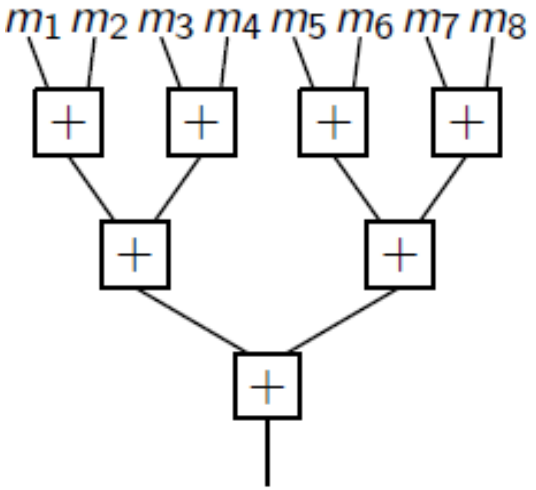
Idee

- in Schaltnetzen niemals alles nacheinander tun.
- Was geht gleichzeitig?

5.3 Multiplikation

Addition vieler Zahlen

Besserer Ansatz : paarweise addieren



Bewertung

- Anzahl Addierer $= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1 \approx n$
- Gesamtgröße ≈ n · Größe(Addierer)
- Tiefe auf i -ter Ebene $n^{log_2(n)-1}$ Addierer
- also $\approx log_2(n)$ Ebenen

Gesamttiefe

$$\approx log_2(n) \cdot 2log_2(n) = 2(log_2\, n)^2$$

5.3 Multiplikation

Addition vieler Zahlen

Gibt es einen noch besseren Ansatz?

Beobachtung

- Addition ersetzt zwei Zahlen
- durch eine Zahl gleicher Summe

zentral für uns

- gleiche Summe → Korrektheit
- weniger Zahlen → Fortschritt

(verrückte?) Idee:

Vielleicht ist es einfacher, drei Zahlen zu ersetzen durch zwei Zahlen gleicher Summe?

5.3 Multiplikation

Addition vieler Zahlen

Gibt es einen noch besseren Ansatz?

(verrückte?) Idee:

Vielleicht ist es einfacher, drei Zahlen zu ersetzen durch zwei Zahlen gleicher Summe?

Beobachtung

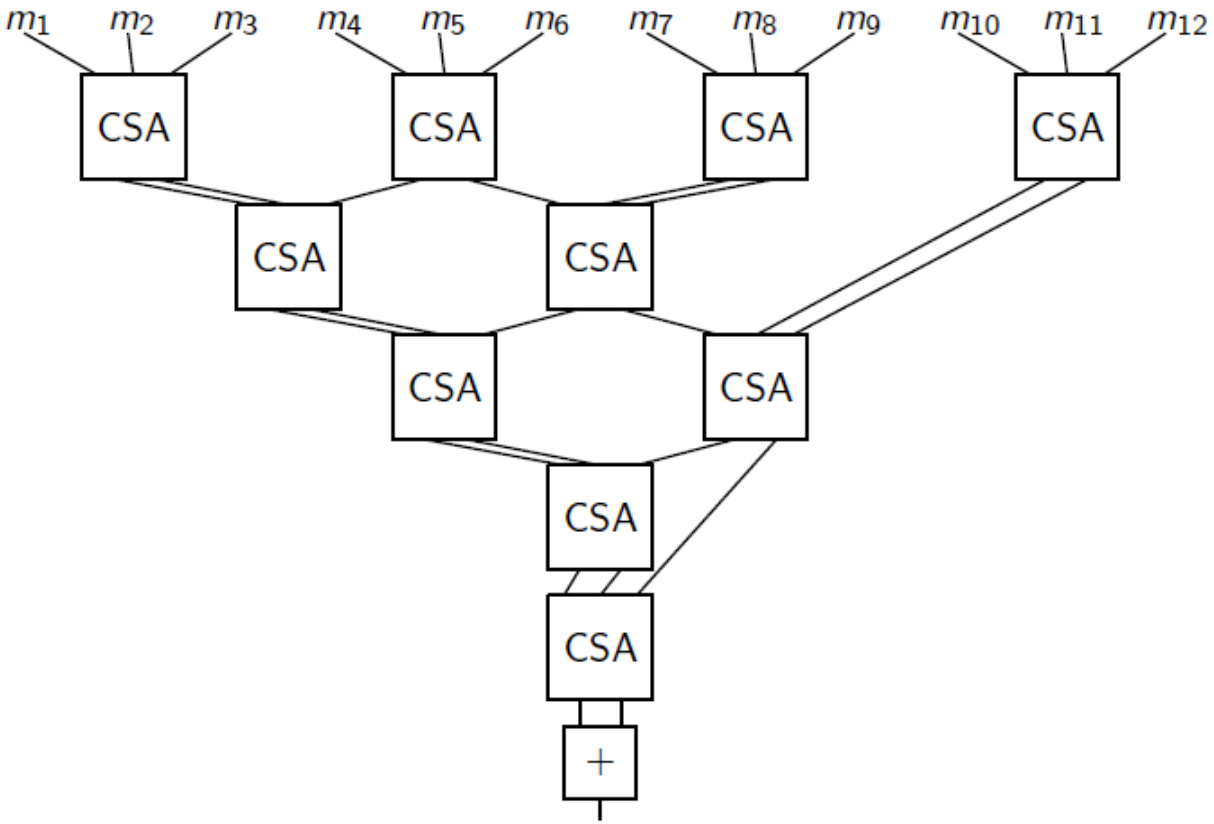
- **es gilt immer noch**
  gleiche Summe → Korrektheit
- **weniger Fortschritt**
  3 → 2 statt
  2 → 1

5.3 Multiplikation

Addition vieler Zahlen

Wallace-Tree

- Carry Save Adder CSA
- m_i sind n-Bit Zahlen
- sinnvoll, wenn CSA flacher ist, als CLA-Addierer



5.3 Multiplikation

Carry Save Adder

Gesucht

- Addierer, mit drei Eingängen und zwei Ausgängen für die gilt:

$$x + y + z = a + b$$

Gefunden

- **Volladierer**, der für $x, y, z \in \{0, 1\}$ folgendes Resultat liefert

$$x + y + z = 2 \cdot c + s$$

- s ist das Summenbit
- $2 \cdot c$ ist das Übertragsbit, bezogen auf seine Stelle 1 Position weiter links

5.3 Multiplikation

Carry Save Adder

- Parallelschaltung von n Volladdierern
- liefern jeweils die **Summenbits**
- und die **Übertragsbits**

Diese Volladdierer sind nicht verkettet

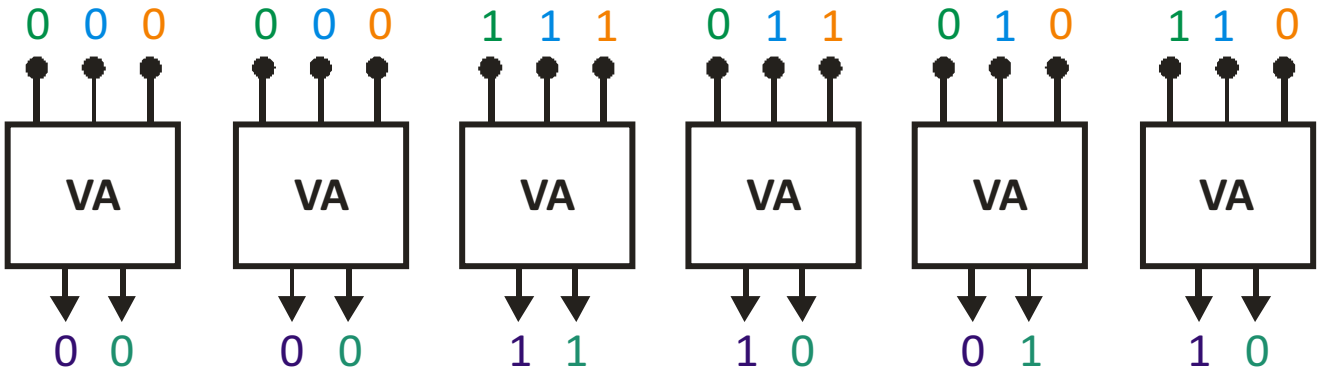
- jeder liefert ein s-Bit und ein c-Bit
- Aus diesen Bits werden
  - das Summen-Wort gebildet, das an der höchst signifikanten Stelle (Anfang) mit 0 erweitert wird
  - und das Carry-Wort, das an der wenigsten signifikanten Stelle (Ende) mit 0 erweitert wird
- **Summe** aus Summen-Wort und Carry-Wort **bildet das Resultat**.

5.3 Multiplikation

Carry Save Adder – Beispiel

Wir addieren

- x = 001001
- y = 001111
- z = 001100



Summen-Wort = (0) 0 0 1 0 1 0 = 0001010

Carry-Wort = 0 0 1 1 0 1 (0) = 0011010

5.3 Multiplikation

Carry Save Adder – Beispiel

Summand x			x_5	x_4	x_3	x_2	x_1	x_0
Summand y	+		y_5	y_4	y_3	y_2	y_1	y_0
Summand z	+		z_5	z_4	z_3	z_2	z_1	z_0
Resultat			$2c_5 + s_5$	$2c_4 + s_4$	$2c_3 + s_3$	$2c_2 + s_2$	$2c_1 + s_1$	$2c_0 + s_0$
Carry-Wort		c_5	c_4	c_3	c_2	c_1	c_0	(0)
Summen-Wort	+	(0)	s_5	s_4	s_3	s_2	s_1	s_0

Summand x			0	0	1	0	0	1
Summand y	+		0	0	1	1	1	1
Summand z	+		0	0	1	1	0	0
Carry-Wort		0	0	1	1	0	1	(0)
Summen-Wort	+	(0)	0	0	1	0	1	0
Summe		0	1	0	0	1	0	0

5.3 Multiplikation

Carry Save Adder

Berechnung von drei n-Bit Zahlen

- Zum Einsatz kommen n Volladdierer
- Größe: $5n$
- Tiefe: 3

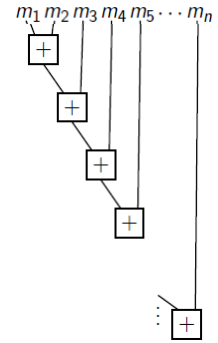
Beobachtung:

- Der Carry (große) Look-Ahead Addierer wird in einem Wallace-Tree erst als letzte Operation benötigt
- Der Wallace-Tree hat somit für die Addition von n Zahlen folgende Eigenschaften
  - Größe: $\sim 5n^2$ (bedingt durch den CLA)
  - Tiefe: $\approx 3log_{3/2}(n) + 2log_2(n) \approx 7.13log_2(n)$

Fazit: Multiplikation ist **wesentlich teurer** als die Addition, aber **nicht wesentlich langsamer**

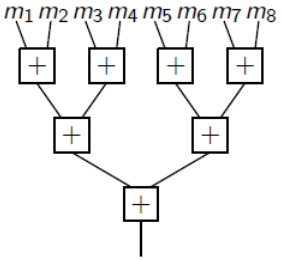
5.3 Multiplikation

Addierer für viele Zahlen



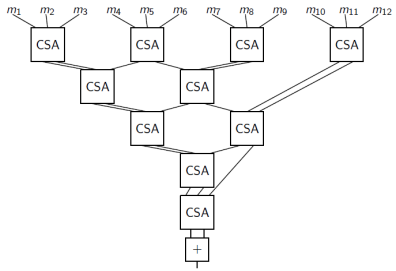
Serielle Addierer

- Tiefe = $(n-1) \cdot$ Tiefe(Addierer)
- Tiefe = $(n-1) \cdot 2log_2(n)$



Paarweise Addierer

- Tiefe =$log_2(n) \cdot$ Tiefe(Addierer)
- Tiefe =$log_2(n) \cdot 2log_2(n) = 2(log_2(n))^2$



Wallace-Tree mit Carry-Save Addierern

- Tiefe: $\approx 3log_{3/2}(n) + 2log_2(n) \approx 7.13log_2(n)$

5. Rechnerarithmetik

1. Einleitung ✓
2. Addition natürlicher Zahlen ✓
3. Multiplikation natürlicher Zahlen ✓
4. **Addition ganzer Zahlen**
5. Addition von Fließkommazahlen
6. Multiplikation von Fließkommazahlen

5.4 Addition ganzer Zahlen

Addition positiver, ganzer Zahlen

- Einerkomplement
- Zweierkomplement
- Vorzeichenbetragdarstellung

- → Addition einfach, da Zahlen binär dargestellt werden, evtl. Überträge beachten

Addition von Zahlen in Exzessdarstellung

- für Exzessdarstellung funktioniert Addierer selbst bei positiven Zahlen nicht

$$x + y = (b + x) + (b + y) = (b + x + y) + b$$

→ Exzessdarstellung fürs Rechnen weitgehend ungeeignet, nur günstig für Vergleiche

5.4 Addition ganzer Zahlen

Warum ist die Addition ganzer Zahlen wichtig?

Beobachtung Niemand muss subtrahieren!

- Statt ”x − y“ einfach ”x + (−y)“ rechnen!
- Idee Ersetze Subtraktion durch

  1. Vorzeichenwechsel
  2. **Addition** einer eventuell negativen Zahl

Bleibt noch zu untersuchen

1. Wie schwierig ist der Vorzeichenwechsel?
2. Wie funktioniert die Addition von negativen Zahlen?

5.4 Addition ganzer Zahlen

Vorzeichenwechsel

Repräsentation	Vorgehen	Kommentar
Vorzeichen-Betrag	Vorzeichen-Bit invertieren	sehr einfach
Einerkomplement	alle Bits invertieren	einfach
Zweierkomplement	alle Bits invertieren, 1 addieren	machbar

Grundsätzlich ist ein Vorzeichenwechsel machbar

5.4 Addition ganzer Zahlen

Addition negativer, ganzer Zahlen

Beobachtung

- Ripple-Carry Addierer
- Carry Look-Ahead Addierer
- sind für Betragszahlen entworfen worden

Frage: Muss die gesamte Hardware für die Addition negativer, ganzer Zahlen neu entworfen werden?

- **Exzessdarstellung** Betrachten wir gar nicht, weil wir damit nicht einmal addieren können.
- **Vorzeichen-Betrag** positive und negative fast gleich dargestellt, darum neuer Schaltnetzentwurf erforderlich

5.4 Addition ganzer Zahlen

Addition negativer Zahlen im Einerkomplement

Auf dieser und nächster Folie

- Notation $\bar{Y}$ ist Komplement von y
- Wir wechseln frei zwischen Zahlen und ihren Repräsentationen.
- feste Darstellungslänge ℓ

Beobachtung $y + \bar{Y} = 2^l - 1$

$\Leftrightarrow$ $\bar{Y} = 2^l - 1 - y$

Rechnung

$$x - y = x + (-y) = x + \bar{Y} = x + 2^l - 1 - y = 2^l + (x - y) - 1$$

Beobachtung

- Darstellungslänge $\Rightarrow 2^l$ „passt nicht“ (Überlauf)
- Überlauf ignorieren $\Rightarrow$ noch 1 addieren **Rechnung korrekt**

5.4 Addition ganzer Zahlen

Addition negativer Zahlen im Zweierkomplement

Beobachtung $y + \bar{Y} = 2^l - 1$

$\Leftrightarrow$ $\bar{Y} = 2^l - 1 - y$

Rechnung

$$x - y = x + (-y) = x + \bar{Y} + 1 = x + 2^l - 1 - y + 1 = 2^l + (x - y)$$

Beobachtung

- Darstellungslänge $\Rightarrow 2^l$ „passt nicht“ (Überlauf)
- Überlauf ignorieren $\Rightarrow$ **Rechnung korrekt**
- Addierer rechnet **richtig** auch für negative Zahlen

- **→ Wir benötigen keine neue Hardware, nur ein paar Regeln!**

5.4 Addition ganzer Zahlen

Überträge bei Addition im Zweierkomplement

Wann ist das Ergebnis korrekt und wann nicht darstellbar?

1. Addition zweier positiver Zahlen

- Ergebnis positiv
- kein Überlauf möglich
- **Ergebnis korrekt, wenn es positiv ist**

2. Addition einer positiven und einer negativen Zahl

- Ergebnis kleiner als größte darstellbare Zahl
- Ergebnis größer als kleinste darstellbare Zahl
- **Ergebnis immer korrekt**

3. Addition zweier negativer Zahlen

- Überlauf entsteht (ignorieren)
- Ergebnis muss negativ sein
- **Ergebnis korrekt, wenn es negativ ist**

5. Rechnerarithmetik

1. Einleitung ✓
2. Addition natürlicher Zahlen ✓
3. Multiplikation natürlicher Zahlen ✓
4. Addition ganzer Zahlen ✓
5. **Addition von Fließkommazahlen**
6. Multiplikation von Fließkommazahlen

5.5 Addition von Fließkommazahlen

Fließkommazahlen

Darstellung gemäß IEEE 754-1985

$$x = (-1)^{s_x} \cdot m_x \cdot 2^{e_x}$$

$$y = (-1)^{s_y} \cdot m_y \cdot 2^{e_y}$$

$$z = x + y = (-1)^{s_z} \cdot m_z \cdot 2^{e_z}$$

- s Vorzeichenbit
- m Mantisse (Binärdarstellung, inklusive impliziter 1)
- e Exponent (Exzessdarstellung, $b = 2^{l-1} - 1$)

Beobachtung

- Addition ist einfach wenn $e_x = e_y = e$ und $s_x = s_y = s$ gilt:
- $z = (-1)^s \cdot (m_x + m_y) \cdot 2^e$

5.5 Addition von Fließkommazahlen

Beobachtung

- Addition ist einfach wenn $e_x = e_y = e$ und $s_x = s_y = s$ gilt:
- $z = (-1)^s \cdot (m_x + m_y) \cdot 2^e$

Idee für Algorithmus

1. Ergebnis wird „so ähnlich“ wie Zahl mit größerem Exponenten, darum Mantisse der Zahl mit kleinerem Exponenten anpassen, dabei auch den Exponenten anpassen

2. Mantissen auf jeden Fall addieren, bei unterschiedlichen Vorzeichen dazu eine Mantisse negieren (Zweierkomplement)

3. anschließend normalisieren

5.5 Addition von Fließkommazahlen

Algorithmus zur Addition

1. Falls $e_x < e_y$, dann x und y komplett vertauschen. Größenvergleich bei Exzess-Darstellung einfach!
2. Falls Vorzeichen der Mantissen ungleich, dann Vorzeichen von s_y invertieren und Übergang von y zu $-y$ im Zweierkomplement
3. Mantisse m_y um $e_x - e_y$ Stellen nach rechts verschieben, dadurch ist der Exponenten gedanklich angeglichen.
   Achtung : Kann zum „Verlust"signifikanter Stellen führen!
4. $m_z := m_x + m_y$
   Falls $e_x = e_y$, Vorzeichenwechsel möglich. Dann s_z invertieren.
5. $e_z = e_y$. Ergebnis normalisieren
   Achtung : Bei Mantissen an implizite Einsen denken!

5.5 Addition von Fließkommazahlen

Beispiel Addition von Gleitkommazahlen

x	1	1001	0101	111	0010	0000	0000	0000	0000
y	1	1001	0100	110	0001	1000	0000	0000	0000

$e_x > e_y$, Vorzeichen gleich, also zunächst nur $s_z := 1$

Mantisse m_y um $e_x - e_y = 1$ Stelle nach rechts verschieben

⇨ 0,111000011

Mantissen addieren

	1,	1	1	1	0	0	1		
+	0,	1	1	1	0	0	0	0	1	1
1	0,	1	1	0	0	0	1	0	1	1

Normalisieren

- Komma um 1 Stelle nach links verschieben ⇨ 1,0110001011
- Exponent um 1 vergrößern ⇨ 1001 0110

Z	1	1001	0110	011	0001	0110	0000	0000	0000

5.5 Addition von Fließkommazahlen

Noch ein Beispiel zur Addition von Gleitkommazahlen

x	1	1000	0101	010	0000	0000	0000	0000	0000
y	0	1000	0100	101	1010	0000	0000	0000	0000

Es gilt: $e_x > e_y$

Da $s_x \neq s_y$ muss s_y invertiert werden

Vorzeichenwechsel bei m_y in Zweierkomplementdarstellung

X	1	1000	0101	1,	010	0000	0000	0000	0000	0000
aus	0	1000	0100	01,	101	1010	0000	0000	0000	0000
wird y	1	1000	0100	10,	010	0110	0000	0000	0000	0000

5.5 Addition von Fließkommazahlen

Noch ein Beispiel zur Addition von Gleitkommazahlen (2)

x	1	1000	0101	1	,	010	0000	0000	0000	0000	0000
y	1	1000	0100	10	,	010	0110	0000	0000	0000	0000

jetzt e_y an e_x anpassen, m_y verschieben

x	1	1000	0101	1	,	010	0000	0000	0000	0000	0000
y	1	1000	0101	11	,	001	0011	0000	0000	0000	0000
z	1	1000	0101	100	,	011	0011	0000	0000	0000	0000

Erinnerung „überfließende“ 1 einfach ignorieren

z	1	1000	0101	0	,	011	0011	0000	0000	0000	0000

Normalisieren Komma um zwei Stellen nach rechts verschieben

Exponent zum Ausgleich um zwei verkleinern

z	1	1000	0011	100	1100	0000	0000	0000	0000

5.5 Addition von Fließkommazahlen

Algorithmus zur Addition

Beobachtung:

- Keine separate Subtraktion erforderlich
- Addition negativer Zahlen enthalten durch Zweierkomplement
- Addition kann mit bekannten Addierern ausgeführt werden

Fehlerquellen

- **Rundung**, wenn Berechnungsergebnis zur korrekten Darstellung mehr signifikante Bits in der Mantisse erfordert, als verfügbar
- **Verlust** niederwertiger Bits durch Angleich der Exponenten während der Addition
- Worst Case: $x \gg y$ und $y \neq 0$ aber $x + y = x$

5.5 Addition von Fließkommazahlen

Probleme bei der Addition: Ein Szenario

Gegeben: Folge von n Gleitkommazahlen $[x_i]$ mit $0 \leq i \leq n$
z.B. gespeichert in einem Feld/Array $x[i]$)

Aufgabe: Berechne Summe S ... möglichst exakt
d.h. mit den Möglichkeiten der Gleitkommaarithmetik

Naive Lösung: Direkte Summation, d.h. berechne:

$$S \uparrow = \sum_{i=0}^{n-1} x_i$$

5.5 Addition von Fließkommazahlen

Probleme bei der Addition: Ein Szenario

Gegeben: Folge von n Gleitkommazahlen $[x_i]$ mit $0 \leq i \leq n$
z.B. gespeichert in einem Feld/Array $x[i]$)

Aufgabe: Berechne Summe S ... möglichst exakt
d.h. mit den Möglichkeiten der Gleitkommaarithmetik

Naive Lösung: Direkte Summation, d.h. berechne:

$$S \uparrow = \sum_{i=0}^{n-1} x_i \quad \text{oder lieber} \quad S \downarrow = \sum_{n-1}^{i=0} x_i$$

5.5 Addition von Fließkommazahlen

Probleme bei der Addition: Ein Szenario

Gegeben: Folge von n Gleitkommazahlen $[x_i]$ mit $0 \leq i \leq n$
z.B. gespeichert in einem Feld/Array $x[i]$)

Aufgabe: Berechne Summe S ... möglichst exakt
d.h. mit den Möglichkeiten der Gleitkommaarithmetik

Naive Lösung: Direkte Summation, d.h. berechne:

$$S\uparrow = \sum_{i=0}^{n-1} x_i \quad \text{oder lieber} \quad S\downarrow = \sum_{n-1}^{i=0} x_i$$

Theorie/Intuition: Beide Summationen liefern dasselbe Ergebnis!

5.5 Addition von Fließkommazahlen

Probleme bei der Addition: Ein Szenario

Gegeben: Folge von n Gleitkommazahlen $[x_i]$ mit $0 \leq i \leq n$
z.B. gespeichert in einem Feld/Array $x[i]$)

Aufgabe: Berechne Summe S ... möglichst exakt
d.h. mit den Möglichkeiten der Gleitkommaarithmetik

Naive Lösung: Direkte Summation, d.h. berechne:

$$S\uparrow = \sum_{i=0}^{n-1} x_i \quad \text{oder lieber} \quad S\downarrow = \sum_{n-1}^{i=0} x_i$$

~~**Theorie/Intuition:** Beide Summationen liefern dasselbe Ergebnis!~~

Praxis: S ↑ und S ↓ sind i.a. nicht gleich!

5. Rechnerarithmetik

1. Einleitung ✓
2. Addition natürlicher Zahlen ✓
3. Multiplikation natürlicher Zahlen ✓
4. Addition ganzer Zahlen ✓
5. Addition von Fließkommazahlen ✓
6. **Multiplikation von Fließkommazahlen**

5.6 Multiplikation von Fließkommazahlen

Gleitkommazahlen-Arithmetik

Darstellung gemäß IEEE 754-1985

$$x = (-1)^{s_x} \cdot m_x \cdot 2^{e_x}$$

$$y = (-1)^{s_y} \cdot m_y \cdot 2^{e_y}$$

- s Vorzeichenbit
- m Mantisse (Binärdarstellung, inklusive impliziter 1)
- e Exponent (Exzessdarstellung, $b = 2^{l-1} - 1$)

Ergebnis $z = (-1)^{s_z} \cdot m_z \cdot 2^{e_z}$

Vereinfachung Wir ignorieren das Runden.

5.6 Multiplikation von Fließkommazahlen

Multiplikation von Gleitkommazahlen

$$x = (-1)^{s_x} \cdot m_x \cdot 2^{e_x}$$

$$y = (-1)^{s_y} \cdot m_y \cdot 2^{e_y}$$

$$z = x \cdot y = (-1)^{s_z} \cdot m_z \cdot 2^{e_z}$$

Beobachtung $z = (-1)^{s_x \oplus s_y} \cdot (m_x \cdot m_y) \cdot 2^{e_x + e_y}$

Vorgehen

1. $s_z := s_x \oplus s_y$
2. $m_z := m_x \cdot m_y$
   - Multiplikation von Betragszahlen wie gesehen,
   - implizite Einsen nicht vergessen!
3. $e_z := e_x + e_y$
   - Addition, wegen Exzessdarstellung $e_x + e_y - b$ berechnen

5.6 Multiplikation von Fließkommazahlen

Beispiel Multiplikation von Gleitkommazahlen

x	1	1000	0101	101	0000	0000	0000	0000	0000
y	1	1000	0111	110	1000	0000	0000	0000	0000

Vorzeichen $s_z = 1 \oplus 1 = 0$

Exponent Bias ist $2^{l-1} - 1 = (1000\ 0000)_2 - 1$

$$e_z := e_x + e_y - b$$

$$\begin{aligned} e_y - b &= (1000\ 0111)_2 - ((1000\ 0000)_2 - 1) \\ &= (1000\ 0111)_2 - (1000\ 0000)_2 + 1 \\ &= (111)_2 + 1 = (1000)_2 \end{aligned}$$

$$e_x + e_y - b = (1000\ 0101)_2 + (1000)_2 = (1000\ 1101)_2$$

→ $(1000\ 1101)_2$ ist vorläufiger Exponent

5.6 Multiplikation von Fließkommazahlen

Beispiel Multiplikation von Gleitkommazahlen

Mantisse

$$
\begin{array}{rcccccccccc}
 & 1, & 1 & 0 & 1 & \cdot & 1, & 1 & 1 & 0 & 1 \\
\hline
 & 1 & 1 & 0 & 1 & & & & & & \\
 & & 1 & 1 & 0 & 1 & & & & & \\
 & & & 1 & 1 & 0 & 1 & & & & \\
 & & & & 0 & 0 & 0 & 0 & & & \\
+ & & & & & 1 & 1 & 0 & 1 & & \\
\hline
1 & 0, & 1 & 1 & 1 & 1 & 0 & 0 & 1 & &
\end{array}
$$

Normalisieren:

- Komma 1 Stelle nach links
- Exponent zum Ausgleich +1
- implizite Eins streichen

z 0 1000 1110 011 1100 1000 0000 0000 0000

5. Rechnerarithmetik

1. Einleitung ✓
2. Addition natürlicher Zahlen ✓
3. Multiplikation natürlicher Zahlen ✓
4. Addition ganzer Zahlen ✓
5. Addition von Fließkommazahlen ✓
6. Multiplikation von Fließkommazahlen ✓

5 Rechnerarithmetik: Real-world numerical catastrophes

- *Ariane 5 rocket.* Ariane 5 rocket exploded 40 seconds after being launched by European Space Agency on June 4th, 1996. (http://www.youtube.com/watch?v=gp_D8r-2hwk) Maiden voyage after a decade and 7 billion dollars of research and development. Sensor reported acceleration that so was large that it caused an overflow in the part of the program responsible for recalibrating inertial guidance. 64-bit floating point number was converted to a 16-bit signed integer, and the conversion failed. This resulted in a drastic attempt to correct the nonexistent problem, which separated the motors from their mountings, leading to the end of Ariane 5.
- *Patriot missile accident.* On February 25, 1991 an American Patriot missile failed to track and destroy an Iraqi Scud missile. Instead it hit an Army barracks, killing 26 people. The cause was later determined to be an inaccurate calculate caused by measuring time in tenth of a second. Couldn't represent 1/10 exactly since used 24 bit floating point
- *Intel FDIV Bug* Error in Pentium hardwire floating point divide circuit. Discovered by Intel in July 1994, rediscovered and publicized by math professor in September 1994. Intel recall in December 1994 cost $300 million. Another floating point bug discovered in 1997.

Source: http://introcs.cs.princeton.edu/java/91float/

Addition von Fließkommazahlen

Fehlerreduktion: Kahan-Summation

Kahan, William (January 1965), Further remarks on reducing truncation errors, Communications of the ACM 8 (1): 40.

Algorithmus zur numerisch stabileren Berechnung von $S \uparrow= \sum_{i=0}^{n-1} x_i$ durch Kompensation verlorener Bits

```
S = 0;                    /* Summe */
E = 0;                    /* geschaetzter Fehler */
for i = 0 to n-1 {
        Y = x[i] - E;     /* bish. Fehler beruecksichtigen */
        Z = S + Y;        /* neues Summationsergebnis */
        E = (Z - S) - Y; /* neue Fehlerschaetzung */
        S = Z;
        }
```

Addition von Fließkommazahlen

Fehlerreduktion: Kahan-Summation (2)

Veranschaulichung des fehlerkompensierenden Berechnungsablaufs für eine Darstellung mit 6 Stellen

S	E	x[i]	Y	Z*	Z	E	S
10000	0	3.14159	3.14149	10003.14159	10003.1	-0.04159	10003.1
10003.1	-0.04159	2.71828	2.75987	10005.85987	10005.9	0.040130	10005.9

```
S = 0;                      /* Summe */
E = 0;                      /* geschaetzter Fehler */
for i = 0 to n-1 {
        Y = x[i] - E;       /* bish. Fehler beruecksichtigen */
        Z = S + Y;          /* neues Summationsergebnis */
        E = (Z - S) - Y;    /* neue Fehlerschaetzung */
        S = Z;
        }
```

Ohne Kompensation wäre das Endergebnis 10005.8 (exakt: 10005.85987)

Sehr wichtiger Artikel über Fließkommazahlen

David Goldberg (1991): What every computer scientist should know about floating-point arithmetic. ACM Computing Serveys 23(1):5–48.