



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN-Fakultät
Fachbereich Informatik



64-040 Modul InfB-RSB

Rechnerstrukturen und Betriebssysteme

[https://tams.informatik.uni-hamburg.de/
lectures/2023ws/vorlesung/rsb](https://tams.informatik.uni-hamburg.de/lectures/2023ws/vorlesung/rsb)

– Kapitel 4 –

Andreas Mäder



Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Wintersemester 2023/2024



Arithmetik

Addition und Subtraktion

Multiplikation

Division

Höhere Funktionen

Mathematische Eigenschaften

Literatur



Wiederholung: Stellenwertsystem („Radixdarstellung“)

- ▶ Wahl einer geeigneten Zahlenbasis b („Radix“)
 - ▶ 10: Dezimalsystem
 - ▶ 16: Hexadezimalsystem (Sedezimalsystem)
 - ▶ 2: Dualsystem
- ▶ Menge der entsprechenden Ziffern $\{0, 1, \dots, b - 1\}$
- ▶ inklusive einer besonderen Ziffer für den Wert Null
- ▶ Auswahl der benötigten Anzahl n von Stellen

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

b Basis a_i Koeffizient an Stelle i

- ▶ universell verwendbar, für beliebig große Zahlen

C:

- ▶ Zahlenbereiche definiert in Headerdatei `/usr/include/limits.h`
`LONG_MIN`, `LONG_MAX`, `ULONG_MAX` etc.
- ▶ Zweierkomplement (signed), Ganzzahl (unsigned)
- ▶ die Werte sind plattformabhängig (!)

Java:

- ▶ 16-bit, 32-bit, 64-bit Zweierkomplementzahlen
- ▶ Wrapper-Klassen `Short`, `Integer`, `Long`

```
Short.MAX_VALUE      =      32767
Integer.MIN_VALUE    = -2147483648
Integer.MAX_VALUE    =  2147483647
Long.MIN_VALUE       = -9223372036854775808L
... 
```

- ▶ Werte sind für die Sprache fest definiert

- ▶ funktioniert genau wie im Dezimalsystem
- ▶ Addition mehrstelliger Zahlen erfolgt stellenweise
- ▶ Additionsmatrix

$$\begin{array}{r|l} + & 0 \ 1 \\ \hline 0 & 0 \ 1 \\ 1 & 1 \ 10 \end{array}$$

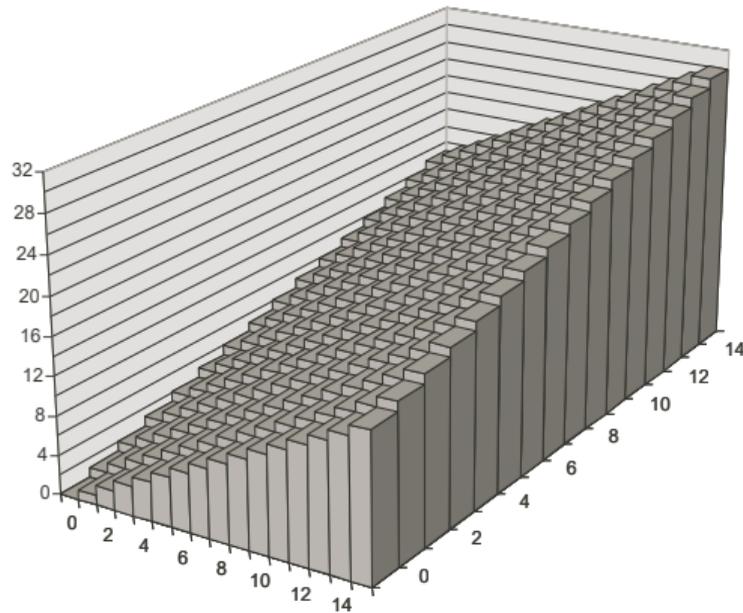
- ▶ Beispiel

$$\begin{array}{r} 1011\ 0011 \\ + 0011\ 1001 \\ \hline \ddot{U} \ 11 \ 11 \\ \hline 1110\ 1100 \end{array} \quad \begin{array}{r} = 179 \\ = 57 \\ \hline 11 \\ \hline = 236 \end{array}$$



unsigned Addition: Visualisierung

Integer addition

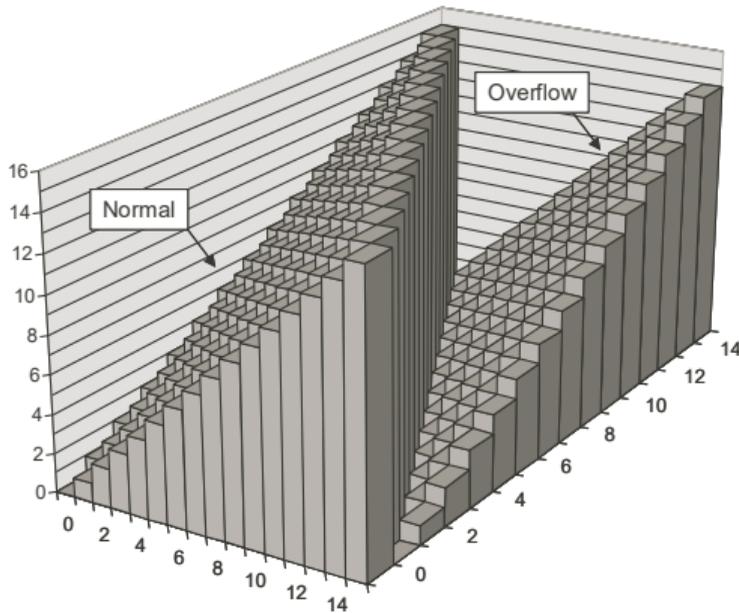


[BO15]

- ▶ Wortbreite der Operanden ist w , hier 4-bit
- ▶ Zahlenbereich der Operanden x, y ist $0 \dots (2^w - 1)$
- ▶ Zahlenbereich des Resultats s ist $0 \dots (2^{w+1} - 2)$

unsigned Addition: Visualisierung (cont.)

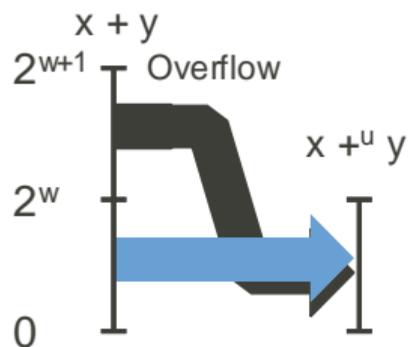
Unsigned addition (4-bit word)



[BO15]

- ▶ Wortbreite der Operanden **und des Resultats** ist w
- ⇒ Überlauf, sobald das Resultat größer als $(2^w - 1)$
- ⇒ oberstes Bit geht verloren

unsigned Addition: Überlauf



- ▶ Wortbreite ist w
- ▶ Zahlenbereich der Operanden x, y ist $0 \dots (2^w - 1)$
- ▶ Zahlenbereich des Resultats s ist $0 \dots (2^{w+1} - 2)$
- ▶ Werte $s \geq 2^w$ werden in den Bereich $0 \dots 2^w - 1$ abgebildet

- ▶ Subtraktion mehrstelliger Zahlen erfolgt stellenweise
- ▶ (Minuend - Subtrahend), Überträge berücksichtigen

- ▶ Beispiel

$$\begin{array}{r} 1011\ 0011 \\ - 0011\ 1001 \\ \hline \text{Ü } 1111 \\ \hline 111\ 1010 \end{array} \quad \begin{array}{r} = 179 \\ = 57 \\ \hline = 122 \end{array}$$

- ▶ Alternative: Subtraktion durch Addition des b -Komplements ersetzen





Subtraktion mit b -Komplement

- ▶ bei Rechnung mit fester Stellenzahl n gilt:

$$K_b(z) + z = b^n = 0$$

weil b^n gerade nicht mehr in n Stellen hineinpasst

- ▶ also gilt für die Subtraktion auch:

$$x - y = x + K_b(y)$$

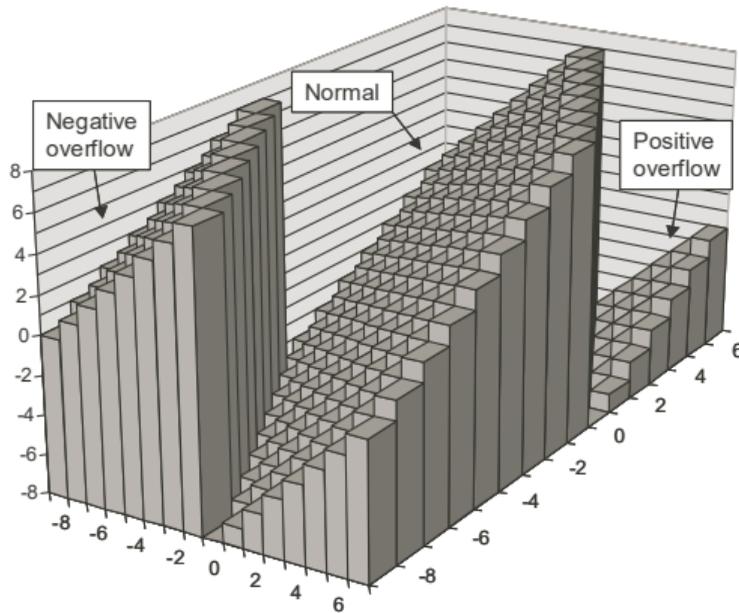
⇒ Subtraktion kann durch Addition des b -Komplements ersetzt werden!

Voraussetzung: begrenzte Stellenanzahl

- ▶ und für Integerzahlen gilt außerdem

$$x - y = x + K_{b-1}(y) + 1$$

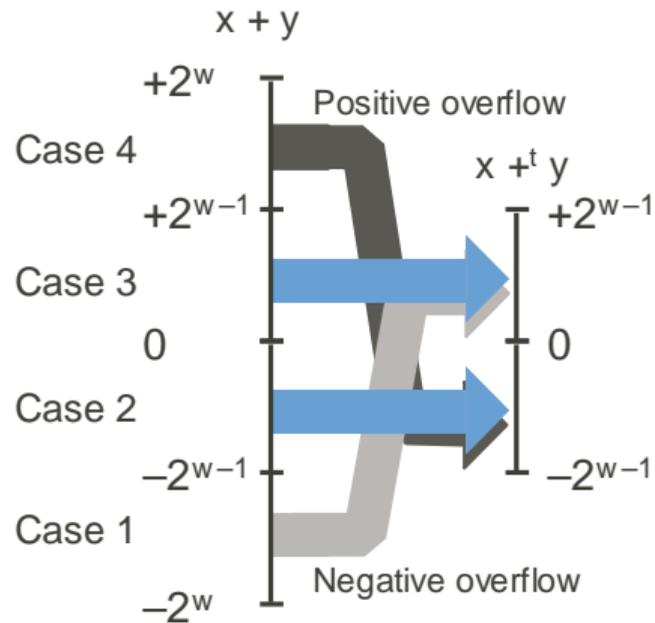
Two's complement addition (4-bit word)



[BO15]

- ▶ Wortbreite der Operanden ist w , hier 4-bit
 - ▶ Zahlenbereich der Operanden x, y ist $-2^{w-1} \dots (2^{w-1} - 1)$
 - ▶ Zahlenbereich des Resultats s ist $-2^w \dots (2^w - 2)$
- ⇒ Überlauf in beide Richtungen möglich

signed Addition: Überlauf



- ▶ Wortbreite des Resultats ist w : Bereich $-2^{w-1} \dots (2^{w-1} - 1)$
- ▶ Überlauf positiv wenn Resultat $\geq 2^{w-1}$: Summe negativ
 -- negativ -- $< -2^{w-1}$: Summe positiv



- ▶ Erkennung eines Überlaufs bei der Addition?
- ▶ wenn beide Operanden das gleiche Vorzeichen haben und sich das Vorzeichen des Resultats unterscheidet

- ▶ Java-Codebeispiel

```
int a, b, sum;           // operands and sum
boolean ovf;            // ovf flag indicates overflow

sum = a + b;
ovf = ((a < 0) == (b < 0)) && ((a < 0) != (sum < 0));
```

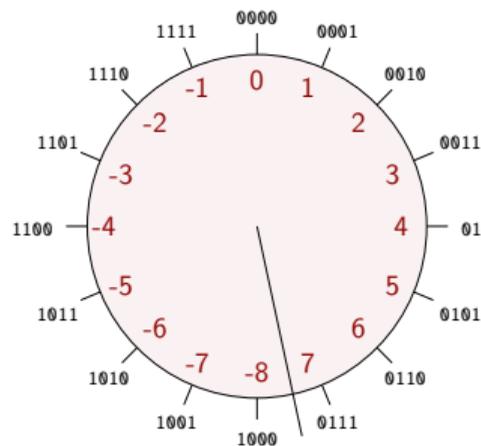
Subtraktion mit Einer- und Zweierkomplement

- ▶ Subtraktion ersetzt durch Addition des Komplements

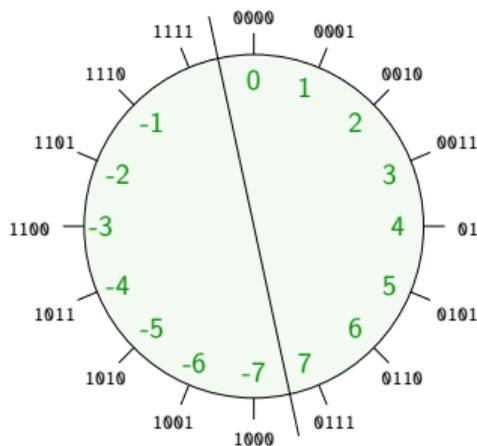
Dezimal	1-Komplement	2-Komplement
$\begin{array}{r} 10 \\ +(-3) \\ \hline +7 \end{array}$	$\begin{array}{r} 0000\ 1010 \\ 1111\ 1100 \\ \hline 1\ 0000\ 0110 \end{array}$	$\begin{array}{r} 0000\ 1010 \\ 1111\ 1101 \\ \hline 1\ 0000\ 0111 \end{array}$
Übertrag:	addieren +1	verwerfen
	$\begin{array}{r} 0000\ 0111 \end{array}$	$\begin{array}{r} 0000\ 0111 \end{array}$

- ▶ **(b-1)-Komplement** der Zahl z $K_{b-1}(z) = b^n - z - b^{-m}$ für $z \neq 0$
 $= 0$ für $z = 0$
- b-Komplement** der Zahl z $K_b(z) = b^n - z$ für $z \neq 0$
 $= 0$ für $z = 0$

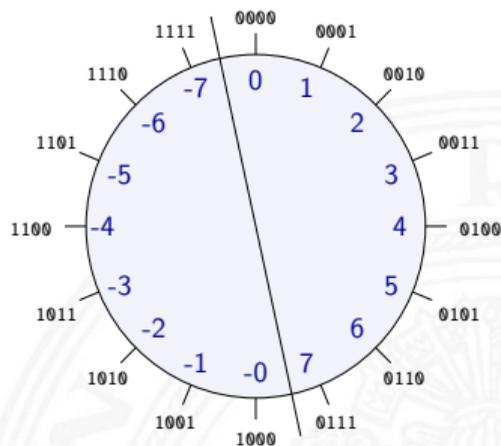
Beispiel für 4-bit Zahlen



2-Komplement



1-Komplement



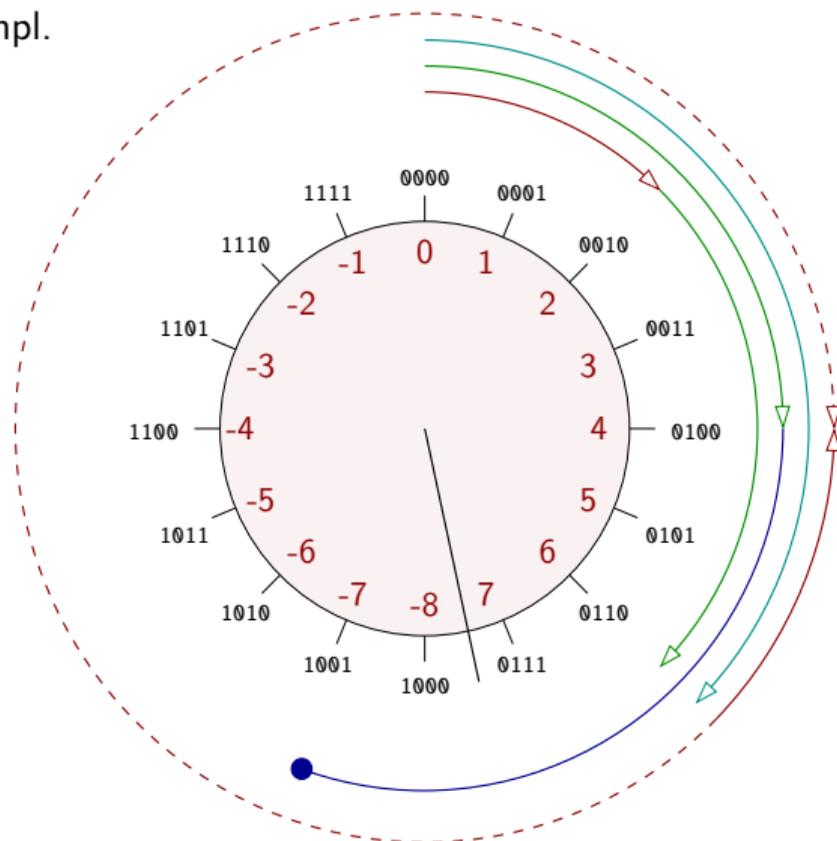
Betrag+Vorzeichen

MSB $\hat{=}$ Vorzeichen

- ▶ Komplement-Arithmetik als Winkeladdition
- ▶ Web-Anwendung: *Visualisierung im Zahlenkreis* (JavaScript, aus [Kor16])

Zahlenkreis: Addition, Subtraktion

2-Kompl.



$$0010+0100=0110$$

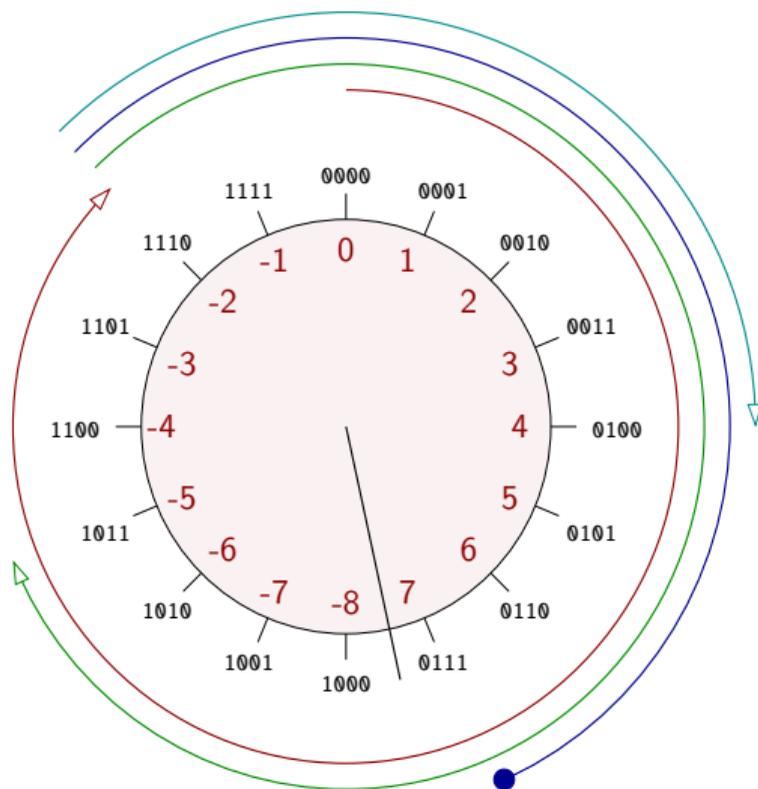
$$0100+0101=1001$$

$$0110-0010=0100$$

0010 1110
0100
0101
0110

Zahlenkreis: Addition, Subtraktion (cont.)

2-Kompl.



$$1110+1101=1011$$

$$1110+1001=0111$$

$$1110+0110=0100$$

1110

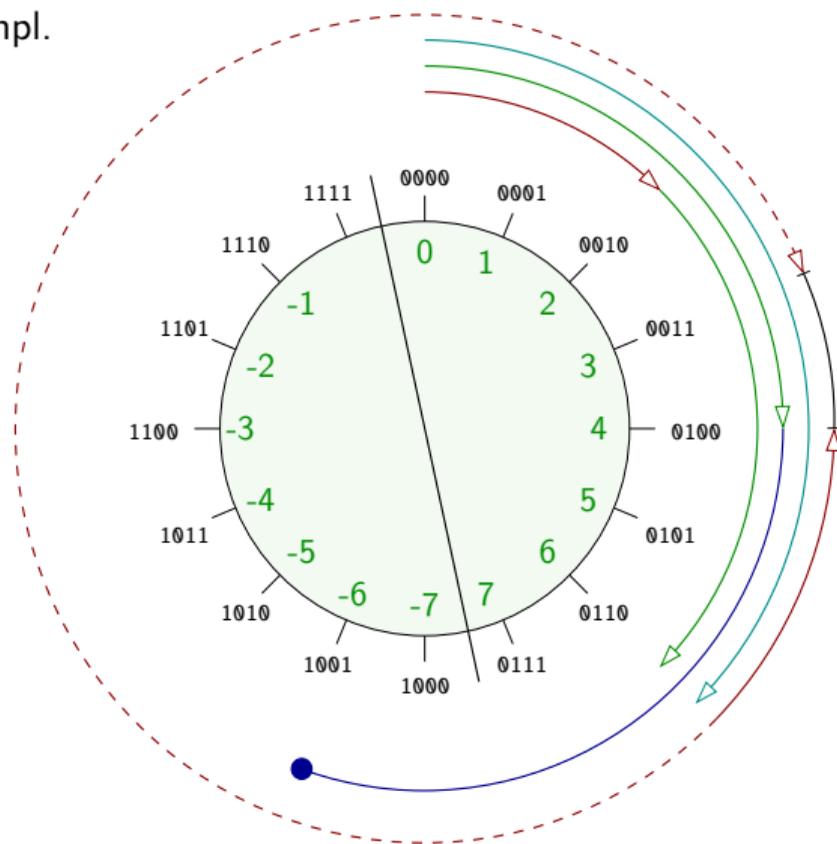
1101

1001

0110

Zahlenkreis: Addition, Subtraktion (cont.)

1-Kompl.



$$0010 + 0100 = 0110$$

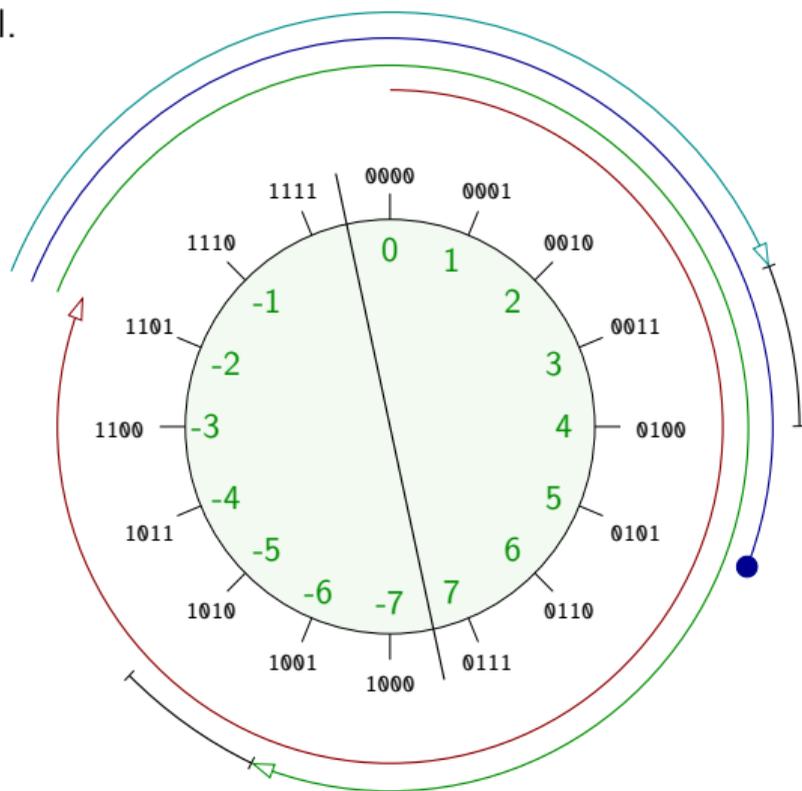
$$0100 + 0101 = 1001$$

$$0110 + 1101 + 1 = 0100$$

0010 1101
0100
0101
0110

Zahlenkreis: Addition, Subtraktion (cont.)

1-Kompl.



$$1101+1100+1=1010$$

$$1101+1000=0101$$

$$1101+0110+1=0100$$

1101

1100

1000

0110



- ▶ für hardwarenahe Programme und Treiber
- ▶ für modulare Arithmetik („multi-precision arithmetic“)
- ▶ aber evtl. ineffizient (vom Compiler schlecht unterstützt)

- ▶ Vorsicht vor solchen Fehlern

```
unsigned int i, cnt = ...;  
for( i = cnt-2; i >= 0; i-- ) {  
    a[i] += a[i+1];  
}
```



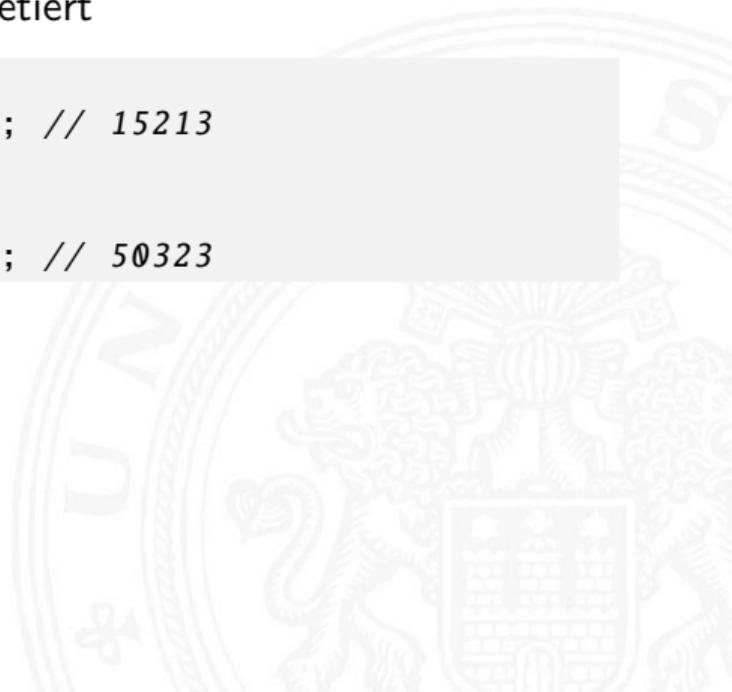


- ▶ Bit-Repräsentation wird nicht verändert
- ▶ kein Effekt auf positiven Zahlen
- ▶ Negative Werte als (große) positive Werte interpretiert

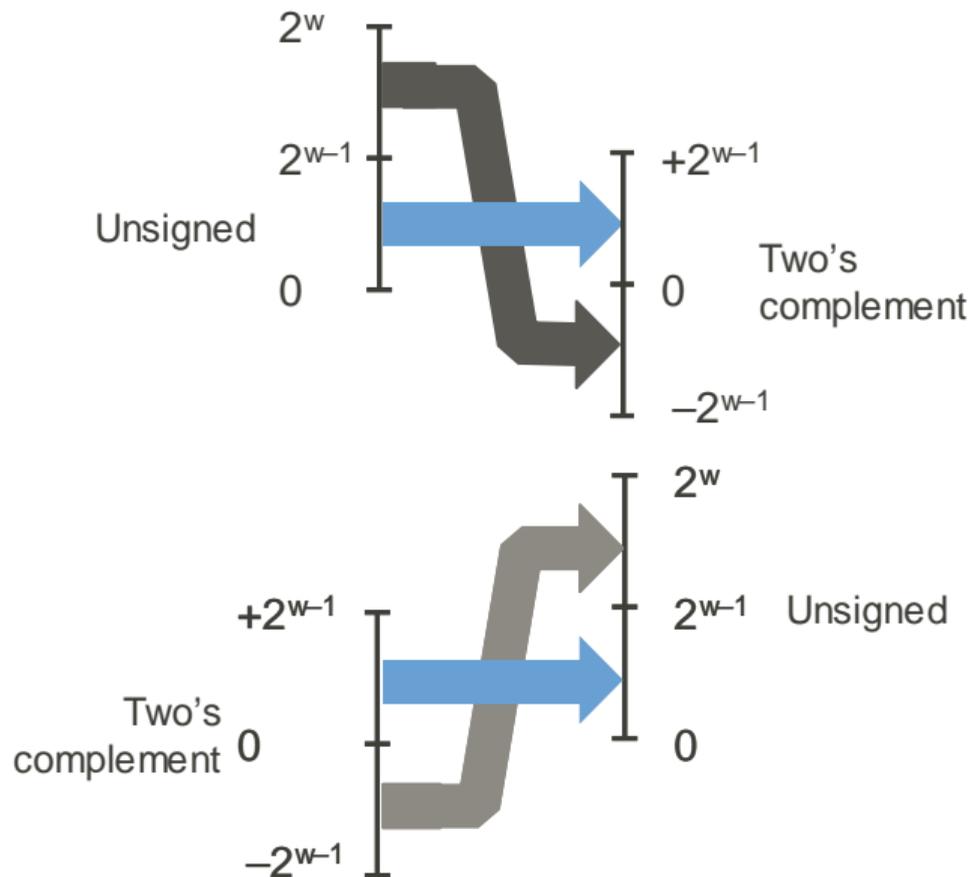
```
short int      x = 15213;
unsigned short int ux = (unsigned short) x; // 15213

short int      y = -15213;
unsigned short int uy = (unsigned short) y; // 50323
```

- ▶ Schreibweise für Konstanten:
 - ▶ ohne weitere Angabe: signed
 - ▶ Suffix „U“ für unsigned: 0U, 4294967259U



in C: unsigned / signed Interpretation



► Arithmetische Ausdrücke:

- bei gemischten Operanden: Auswertung als unsigned
- auch für die Vergleichsoperationen <, >, ==, <=, >=
- Beispiele für Wortbreite 32-bit:

Konstante 1	Relation	Konstante 2	Auswertung	Resultat
0	==	0U	unsigned	1
-1	<	0	signed	1
-1	<	0U	unsigned	0
2147483647	>	-2147483648	signed	1
2147483647U	>	-2147483648	unsigned	0
2147483647	>	(int) 2147483648U	signed	1
-1	>	-2	signed	1
(unsigned) -1	>	-2	unsigned	1

Fehler



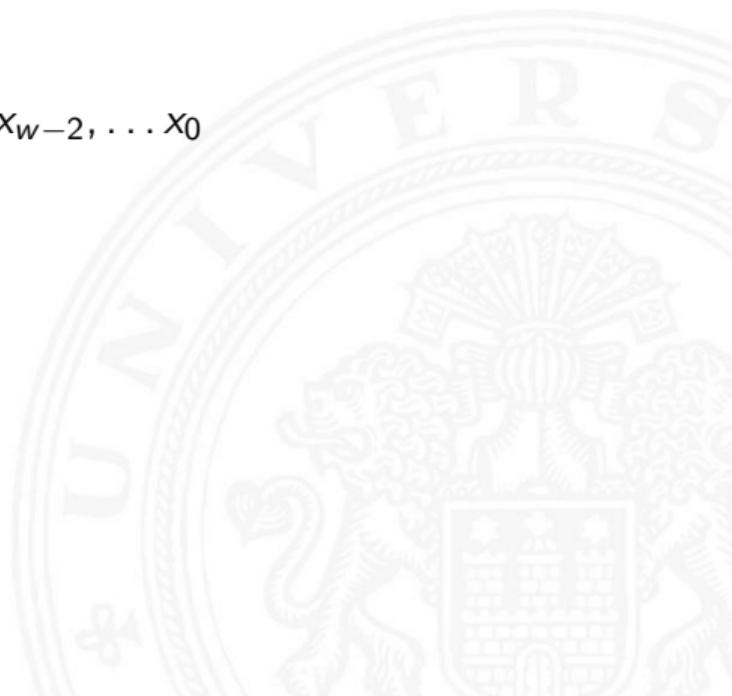
- ▶ Gegeben: w -bit Integer x
- ▶ Umwandeln in $w + k$ -bit Integer x' mit gleichem Wert?

- ▶ **Sign-Extension:** Vorzeichenbit kopieren

$$x' = x_{w-1}, \dots, x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_0$$

- ▶ Zahlenbeispiele

0110	4-bit signed: +6
0000 0110	8-bit signed: +6
0000 0000 0000 0110	16-bit signed: +6
1110	4-bit signed: -2
1111 1110	8-bit signed: -2
1111 1111 1111 1110	16-bit signed: -2





Java Puzzlers No.5

J. Bloch, N. Gafter: *Java Puzzlers: Traps, Pitfalls, and Corner Cases*. Addison-Wesley, 2005

4.1 Arithmetik - Addition und Subtraktion

64-040 Rechnerstrukturen und Betriebssysteme

```
public static void main( String[] args ) {  
    System.out.println(  
        Long.toHexString( 0x100000000L + 0xcafebabe ));  
}
```

- ▶ Programm addiert zwei Konstanten, Ausgabe in Hex-Format
- ▶ Was ist das Resultat der Rechnung?



Java Puzzlers No.5

J. Bloch, N. Gafter: *Java Puzzlers: Traps, Pitfalls, and Corner Cases*. Addison-Wesley, 2005

4.1 Arithmetik - Addition und Subtraktion

64-040 Rechnerstrukturen und Betriebssysteme

```
public static void main( String[] args ) {  
    System.out.println(  
        Long.toHexString( 0x100000000L + 0xcafebabe ));  
}
```

- ▶ Programm addiert zwei Konstanten, Ausgabe in Hex-Format
- ▶ Was ist das Resultat der Rechnung?

0xffffffffcafebabe **sign-extension!**

0x0000000100000000

Ü 11111110

00000000cafebabe

- ▶ Erstflug der Ariane-5 („V88“) am 04. Juni 1996
 - ▶ Kurskorrektur wegen vermeintlich falscher Fluglage
 - ▶ Selbstzerstörung der Rakete nach 36,7 Sekunden
 - ▶ Schaden ca. 635 M€
(teuerster Softwarefehler der Geschichte?)
-
- ▶ bewährte Software von Ariane-4 übernommen
 - ▶ aber Ariane-5 viel schneller als Ariane-4
 - ▶ 64-bit Gleitkommawert für horizontale Geschwindigkeit
 - ▶ Umwandlung in 16-bit Integer: dabei Überlauf
-
- ▶ de.wikipedia.org/wiki/Ariane_V88





Multiplikation im Dualsystem

- ▶ funktioniert genau wie im Dezimalsystem
- ▶ $p = a \cdot b$ mit Multiplikator a und Multiplikand b
- ▶ Multiplikation von a mit je einer Stelle des Multiplikanten b
- ▶ Addition der Teilterme

- ▶ Multiplikationsmatrix – sehr einfach: $\cdot 0 / \cdot 1$

\cdot		0	1
0		0	0
1		0	1





Multiplikation im Dualsystem (cont.)

► Beispiel

$$\begin{array}{r}
 10110011 \cdot 1101 \\
 \hline
 10110011 \quad 1 \\
 10110011 \quad 1 \\
 00000000 \quad 0 \\
 10110011 \quad 1 \\
 \hline
 \text{Ü } 11101111 \\
 \hline
 100100010111
 \end{array}
 = 179 \cdot 13 = 2327$$

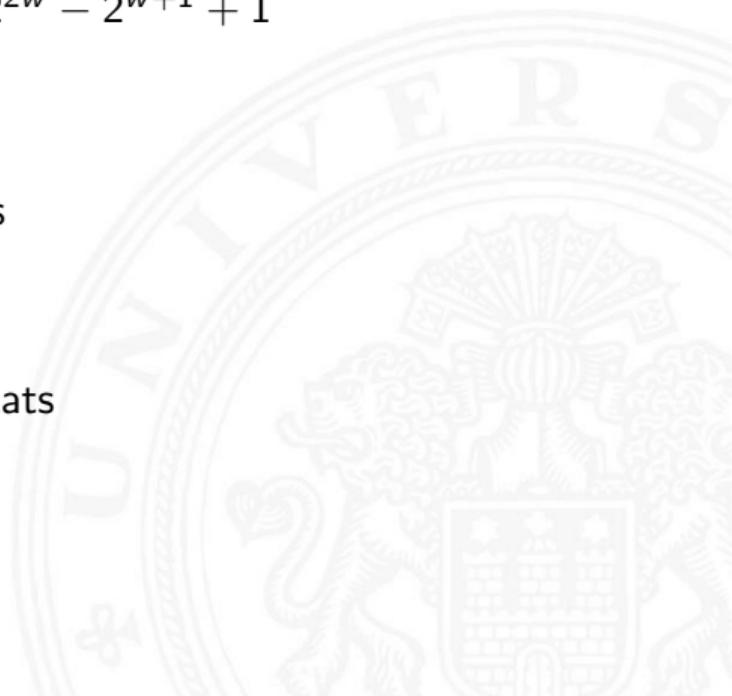
$$\begin{array}{l}
 = 1001\ 0001\ 0111 \\
 = 0x917
 \end{array}$$





- ▶ bei Wortbreite w bit
 - ▶ Zahlenbereich der Operanden: $0 \dots (2^w - 1)$
 - ▶ Zahlenbereich des Resultats: $0 \dots (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
- ⇒ bis zu $2w$ bits erforderlich

- ▶ C: Resultat enthält nur die unteren w bits
- ▶ Java: keine unsigned Integer
- ▶ Hardware: teilweise zwei Register *high*, *low* für die oberen und unteren Bits des Resultats





- ▶ Zahlenbereich der Operanden: $-2^{w-1} \dots (2^{w-1} - 1)$
 - ▶ Zahlenbereich des Resultats: $-2^{w-1} \cdot (2^{w-1} - 1) \dots (2^{2w-2})$
- ⇒ bis zu $2w$ bits erforderlich

- ▶ C, Java: Resultat enthält nur die unteren w bits
- ▶ Überlauf wird ignoriert

```
int i = 100*200*300*400; // -1894967296
```

- ▶ Repräsentation der unteren Bits des Resultats entspricht der unsigned Multiplikation
- ⇒ kein separater Algorithmus erforderlich
Beweis: siehe Bryant, O'Hallaron: Abschnitt 2.3.5 [BO15]



Java Puzzlers No. 3

J. Bloch, N. Gafter: *Java Puzzlers: Traps, Pitfalls, and Corner Cases*. Addison-Wesley, 2005

4.2 Arithmetik - Multiplikation

64-040 Rechnerstrukturen und Betriebssysteme

```
public static void main( String args[] ) {  
    final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;  
    final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;  
    System.out.println( MICROS_PER_DAY / MILLIS_PER_DAY );  
}
```

Java Puzzlers No. 3

J. Bloch, N. Gafter: *Java Puzzlers: Traps, Pitfalls, and Corner Cases*. Addison-Wesley, 2005

4.2 Arithmetik - Multiplikation

64-040 Rechnerstrukturen und Betriebssysteme

```
public static void main( String args[] ) {  
    final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;  
    final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;  
    System.out.println( MICROS_PER_DAY / MILLIS_PER_DAY );  
}
```

- ▶ druckt den Wert 5, nicht 1000!
 - ▶ MICROS_PER_DAY wird mit 32-bit berechnet, dabei Überlauf
 - ▶ Konvertierung nach 64-bit long erst bei der Zuweisung
- ⇒ long-Konstante schreiben: `24L * 60 * 60 * 1000 * 1000`



- ▶ $d = a/b$ mit Dividend a und Divisor b
- ▶ funktioniert genau wie im Dezimalsystem
- ▶ schrittweise Subtraktion des Divisors
- ▶ Berücksichtigen des „Stellenversetzens“
- ▶ in vielen Prozessoren nicht (oder nur teilweise) durch Hardware unterstützt
- ▶ daher deutlich langsamer als Multiplikation





► Beispiele

$$100_{10} / 3_{10} = 110\ 0100_2 / 11_2 = 10\ 0001_2$$

$$\begin{array}{r}
 1100100 \ / \ 11 = 0100001 \\
 \begin{array}{r}
 1 \qquad \qquad \qquad 0 \\
 11 \qquad \qquad \qquad 1 \\
 -11 \\
 \hline
 0 \qquad \qquad \qquad 0 \\
 0 \qquad \qquad \qquad 0 \\
 1 \qquad \qquad \qquad 0 \\
 10 \qquad \qquad \qquad 0 \\
 100 \qquad \qquad \qquad 1 \\
 -11 \\
 \hline
 1 \qquad \qquad \qquad 1 \text{ (Rest)}
 \end{array}
 \end{array}$$





Division im Dualsystem (cont.)

$$91_{10}/13_{10} = 101\ 1011_2/1101_2 = 111_2$$

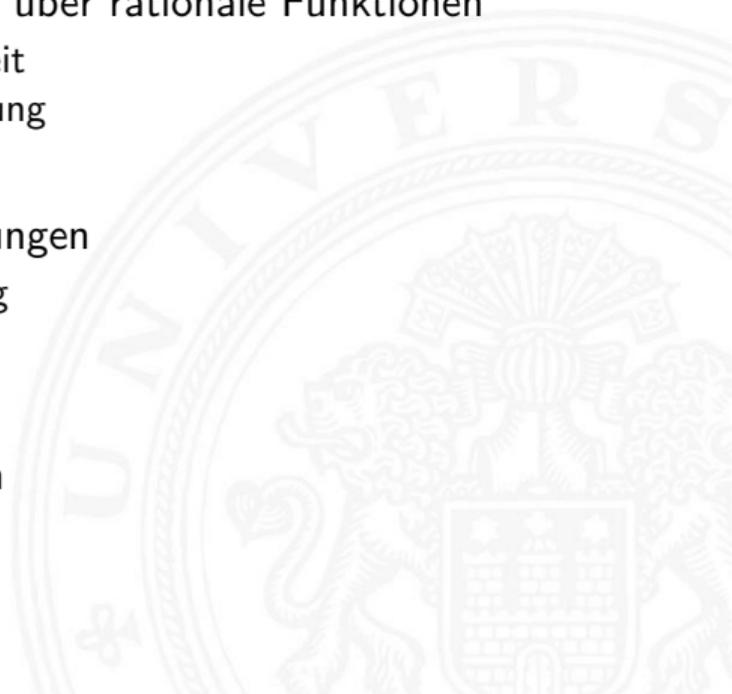
$$\begin{array}{r} 1011011 \ / \ 1101 = 0111 \\ 1011 \qquad \qquad \qquad 0 \\ 10110 \qquad \qquad \qquad 1 \\ \hline -1101 \\ \hline 10011 \qquad \qquad \qquad 1 \\ -1101 \\ \hline 01101 \qquad \qquad \qquad 1 \\ -1101 \\ \hline 0 \end{array}$$





Berechnung von \sqrt{x} , $\log x$, $\exp x$, $\sin x \dots$?

- ▶ Approximation über Polynom (Taylor-Reihe) bzw. über rationale Funktionen
 - ▶ vorberechnete Koeffizienten für höchste Genauigkeit
 - ▶ Ausnutzen mathematischer Identitäten für Skalierung
- ▶ Sukzessive Approximation über iterative Berechnungen
 - ▶ Beispiele: Quadratwurzel und Reziprok-Berechnung
 - ▶ häufig schnelle (quadratische) Konvergenz
- ▶ Berechnungen erfordern nur die Grundrechenarten





- ▶ Berechnung des Reziprokwerts $y = 1/x$ über

$$y_{i+1} = y_i \cdot (2 - x \cdot y_i)$$

- ▶ geeigneter Startwert y_0 als Schätzung erforderlich

- ▶ Beispiel $x = 3$, $y_0 = 0,5$:

$$\begin{aligned} y_1 &= 0,5 \cdot (2 - 3 \cdot 0,5) &&= 0,25 \\ y_2 &= 0,25 \cdot (2 - 3 \cdot 0,25) &&= 0,3125 \\ y_3 &= 0,3125 \cdot (2 - 3 \cdot 0,3125) &&= 0,33203125 \\ y_4 &= 0,3332824 \\ y_5 &= 0,3333333332557231 \\ y_6 &= 0,3333333333333333 \end{aligned}$$





Quadratwurzel: Heron-Verfahren für \sqrt{x}

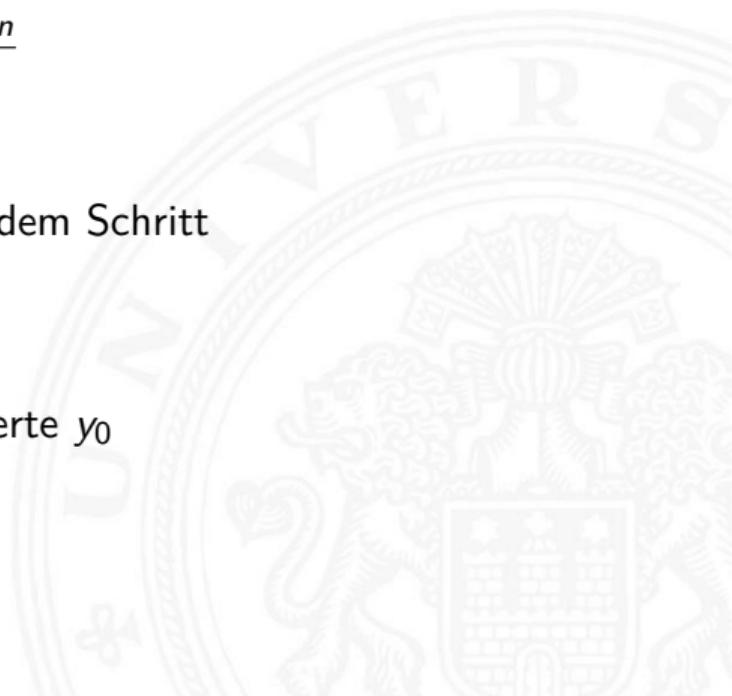
Babylonisches Wurzelziehen

- ▶ Sukzessive Approximation von $y = \sqrt{x}$ gemäß

$$y_{n+1} = \frac{y_n + x/y_n}{2}$$

- ▶ quadratische Konvergenz in der Nähe der Lösung
- ▶ Anzahl der gültigen Stellen verdoppelt sich mit jedem Schritt

- ▶ aber langsame Konvergenz fernab der Lösung
- ▶ Lookup-Tabelle und Tricks für brauchbare Startwerte y_0





Welche mathematischen Eigenschaften gelten bei der Informationsverarbeitung / in der gewählten Repräsentation?

Beispiele:

▶ Gilt $x^2 \geq 0$?

- ▶ float: ja
- ▶ signed integer: nein

▶ Gilt $(x + y) + z = x + (y + z)$?

- ▶ integer: ja
- ▶ float: nein

$$1.0E20 + (-1.0E20 + 3.14) = 0$$





unsigned Arithmetik

- ▶ Wortbreite auf w begrenzt
- ▶ kommutative Gruppe / Abel'sche Gruppe
 - ▶ Abgeschlossenheit $0 \leq a \oplus_w^u b \leq 2^w - 1$
 - ▶ Kommutativgesetz $a \oplus_w^u b = b \oplus_w^u a$
 - ▶ Assoziativgesetz $a \oplus_w^u (b \oplus_w^u c) = (a \oplus_w^u b) \oplus_w^u c$
 - ▶ neutrales Element $a \oplus_w^u 0 = a$
 - ▶ Inverses $a \oplus_w^u \bar{a} = 0; \bar{a} = 2^w - a$



signed Arithmetik

2-Komplement

- ▶ Wortbreite auf w begrenzt
- ▶ signed und unsigned Addition sind auf Bit-Ebene identisch

$$a \oplus_w^s b = U2S(S2U(a) \oplus_w^u S2U(b))$$

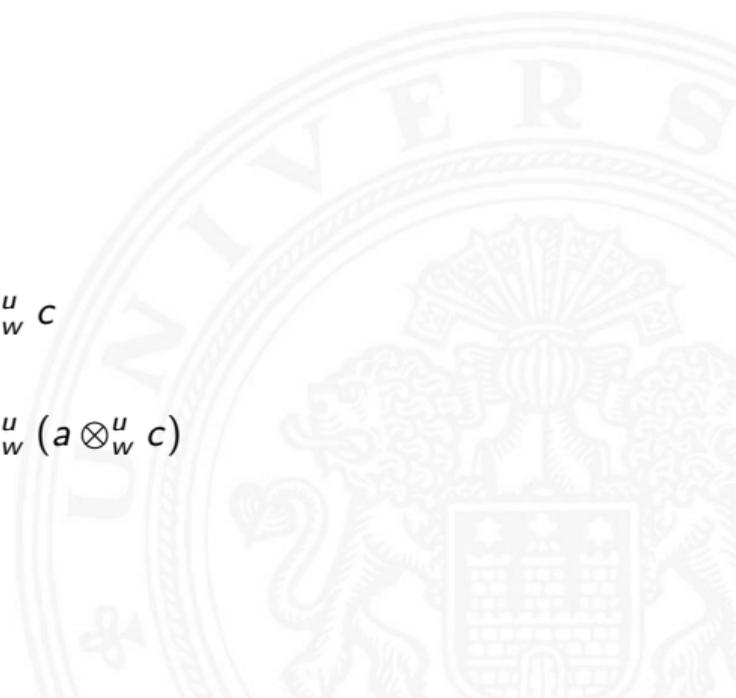
⇒ isomorphe Algebra zu \oplus_w^u

- ▶ kommutative Gruppe / Abel'sche Gruppe
 - ▶ Abgeschlossenheit $-2^{w-1} \leq a \oplus_w^s b \leq 2^{w-1} - 1$
 - ▶ Kommutativgesetz $a \oplus_w^s b = b \oplus_w^s a$
 - ▶ Assoziativgesetz $a \oplus_w^s (b \oplus_w^s c) = (a \oplus_w^s b) \oplus_w^s c$
 - ▶ neutrales Element $a \oplus_w^s 0 = a$
 - ▶ Inverses $a \oplus_w^s \bar{a} = 0; \quad \bar{a} = -a, a \neq -2^{w-1}$
 $a, a = -2^{w-1}$



unsigned Arithmetik

- ▶ Wortbreite auf w begrenzt
- ▶ Modulo-Arithmetik $a \otimes_w^u b = (a \cdot b) \bmod 2^w$
- ▶ \otimes_w^u und \oplus_w^u bilden einen kommutativen Ring
 - ▶ \oplus_w^u ist eine kommutative Gruppe
 - ▶ Abgeschlossenheit $0 \leq a \otimes_w^u b \leq 2^w - 1$
 - ▶ Kommutativgesetz $a \otimes_w^u b = b \otimes_w^u a$
 - ▶ Assoziativgesetz $a \otimes_w^u (b \otimes_w^u c) = (a \otimes_w^u b) \otimes_w^u c$
 - ▶ neutrales Element $a \otimes_w^u 1 = a$
 - ▶ Distributivgesetz $a \otimes_w^u (b \oplus_w^u c) = (a \otimes_w^u b) \oplus_w^u (a \otimes_w^u c)$





signed Arithmetik

- ▶ signed und unsigned Multiplikation sind auf Bit-Ebene identisch
- ▶ ...

isomorphe Algebren

- ▶ unsigned Addition und Multiplikation; Wortbreite w
- ▶ signed Addition und Multiplikation; Wortbreite w
- ▶ isomorph zum Ring der ganzen Zahlen *modulo* 2^w

- ▶ Ordnungsrelation im Ring der ganzen Zahlen

- ▶ $a > 0 \quad \longrightarrow \quad a + b > b$

- ▶ $a > 0, b > 0 \longrightarrow a \cdot b > 0$

- ▶ diese Relationen **gelten nicht** bei Rechnerarithmetik!

2-Komplement

Überlauf

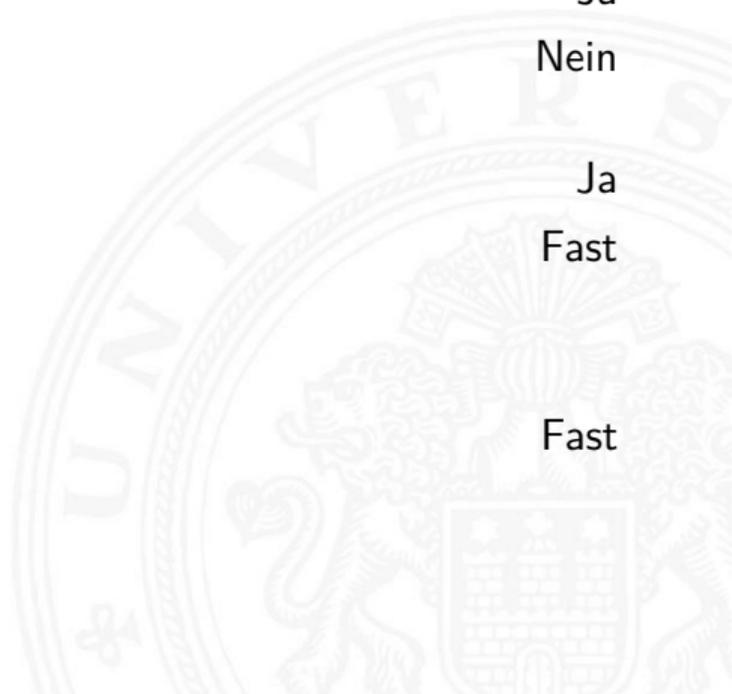


Gleitkomma Addition

Vergleich mit kommutativer Gruppe

- ▶ Abgeschlossen? Ja
- ▶ Kommutativ? Ja
- ▶ Assoziativ? Nein
(Überlauf, Rundungsfehler)
- ▶ Null ist neutrales Element? Ja
- ▶ Inverses Element existiert? Fast
(außer für NaN und Infinity)

- ▶ Monotonie? $a \geq b \longrightarrow (a + c) \geq (b + c)$
(außer für NaN und Infinity) Fast



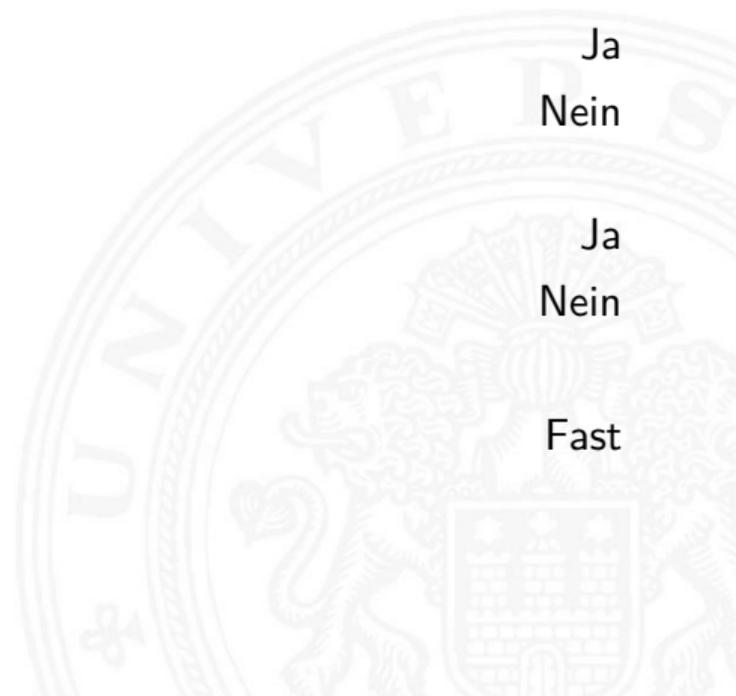


Gleitkomma Multiplikation

Vergleich mit kommutativem Ring

- ▶ Abgeschlossen?
(aber Infinity oder NaN möglich) Ja
- ▶ Kommutativ? Ja
- ▶ Assoziativ?
(Überlauf, Rundungsfehler) Nein
- ▶ Eins ist neutrales Element? Ja
- ▶ Distributivgesetz? Nein

- ▶ Monotonie? $a \geq b; c \geq 0 \longrightarrow (a \cdot c) \geq (b \cdot c)$
(außer für NaN und Infinity) Fast





[BO15] R.E. Bryant, D.R. O'Hallaron:

Computer systems – A programmers perspective.

3rd global ed., Pearson Education Ltd., 2015.

csapp.cs.cmu.edu

ISBN 978-1-292-10176-7

[TA14] A.S. Tanenbaum, T. Austin:

Rechnerarchitektur – Von der digitalen Logik zum Parallelrechner.

6. Auflage, Pearson Deutschland GmbH, 2014.

ISBN 978-3-8689-4238-5

[Kor16] Laszlo Korte: *TAMS Tools for eLearning.*

Uni Hamburg, FB Informatik, 2016, BSc Thesis.

tams.informatik.uni-hamburg.de/research/software/tams-tools



[Omo94] A.R. Omondi: *Computer Arithmetic Systems – Algorithms, Architecture and Implementations*.
Prentice-Hall International, 1994.

ISBN 978-0-13-334301-4

[Kor01] I. Koren: *Computer Arithmetic Algorithms*.
2nd edition, CRC Press, 2001.
www.ecs.umass.edu/ece/koren/arith

ISBN 978-1-568-81160-4

[Spa76] O. Spaniol: *Arithmetik in Rechenanlagen*.
B. G. Teubner, 1976.

ISBN 978-3-519-02332-6

