ulm university universität ulm

Grundlagen der Betriebssysteme

[CS2100]

Sommersemester 2014

Heiko Falk

Institut für Eingebettete Systeme/Echtzeitsysteme
Ingenieurwissenschaften und Informatik
Universität Ulm

ulm university universität ulm

Kapitel 2

Zahlendarstellungen und Rechnerarithmetik

Inhalte der Vorlesung

1. Einführung
2. **Zahlendarstellungen und Rechnerarithmetik**
3. Einführung in Betriebssysteme
4. Prozesse und Nebenläufigkeit
5. Filesysteme
6. Speicherverwaltung
7. Einführung in MIPS-Assembler
8. Rechteverwaltung
9. Ein-/Ausgabe und Gerätetreiber

Inhalte des Kapitels

2. Zahlendarstellungen und Rechnerarithmetik

- Natürliche Zahlen
  - Darstellung zur Basis *b*, Umrechnung
  - Relevante Darstellungen: binär, oktal, dezimal, hexadezimal
- Binäre Arithmetik
  - Boolesche Algebra
  - Addition: Halbaddierer, Volladdierer
  - Subtraktion: Zweierkomplement-Darstellung
  - Multiplikation: Booth-Algorithmus
  - Division: „Schulmethode“
- Reelle Zahlen
  - Festkommazahlen
  - Gleitkommazahlen: Darstellung als Mantisse und Exponent
  - IEEE 754: Gleitkomma-Darstellung, spezielle Werte
- Zeichensätze

Positive Ganze Zahlen (1)

Positionale Zahlendarstellung

- Ziffern
- Position der Ziffern gewichtet ihren Wert

Dezimalsystem

- Beispiel:

  $4711 = (4, 7, 1, 1)_{10} = 4 * 10^3 + 7 * 10^2 + 1 * 10^1 + 1 * 10^0$

- Allgemein: *n*-stellige Dezimalzahl

  $(z_{n-1}, z_{n-2}, ..., z_2, z_1, z_0)_{10} =$

  $z_{n-1} * 10^{n-1} + z_{n-2} * 10^{n-2} + ... + z_2 * 10^2 + z_1 * 10^1 + z_0 * 10^0$

  mit $z_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Positive Ganze Zahlen (2)

Dualsystem, Binärsystem

– Beispiel:

$1011_2 = (1, 0, 1, 1)_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

– Allgemein: *n*-stellige Binärzahl

$(z_{n-1}, z_{n-2}, ..., z_2, z_1, z_0)_2 =$

$z_{n-1} * 2^{n-1} + z_{n-2} * 2^{n-2} + ... + z_2 * 2^2 + z_1 * 2^1 + z_0 * 2^0$

mit $z_i \in \{0, 1\}$

Positive Ganze Zahlen (3)

Allgemein

– Darstellung Natürlicher Zahlen durch Zahlensystem zu einer beliebigen Basis $b \geq 1$

$(z_{n-1}, z_{n-2}, ..., z_2, z_1, z_0)_b =$

$z_{n-1} * b^{n-1} + z_{n-2} * b^{n-2} + ... + z_2 * b^2 + z_1 * b^1 + z_0 * b^0$

mit $z_i \in \{0, 1, ..., b\text{-}1\}$

Typische Basen für Rechnerarithmetik

– $b = 2$ Binärsystem

– $b = 8$ Oktalsystem $z_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$

– $b = 10$ Dezimalsystem

– $b = 16$ Hexadezimalsystem

$z_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Konvertierung der Darstellung (1)

Umwandlung von einer Zahlendarstellung in die andere

- Basis des Ziel-Zahlensystems als Divisor
- Reste bilden die Ziffern der Darstellung

- Beispiel: $19_{10} \rightarrow x_2$

$19 \div 2 = 9$	Rest 1	*least significant digit*
$9 \div 2 = 4$	Rest 1	
$4 \div 2 = 2$	Rest 0	
$2 \div 2 = 1$	Rest 0	
$1 \div 2 = 0$	Rest 1	*most significant digit*

$x = (1, 0, 0, 1, 1)_2$

Konvertierung der Darstellung (2)

Alternative

- Finden der Stufenzahlen und deren Vielfachen

- Beispiel: $19_{10} \rightarrow x_8$
  - Stufenzahlen des Oktalsystems: $8^0 = 1$, $8^1 = 8$, $8^2 = 64$, $8^3 = 512$, ...
  - Welche Stufenzahl passt so gerade noch hinein: 8
  - Wie oft passt sie hinein:
    $19 \div 8 = 2$ Rest 3
  - Wiederholung mit Rest und nächst kleinerer Stufenzahl:
    $3 \div 1 = 3$ Rest 0
  - Wiederholung bis Stufenzahl 1 erreicht wurde
  - Ergebnisse der Division bilden die Ziffern:
    $x = 23_8$

Roter Faden

2. Zahlendarstellungen und Rechnerarithmetik

- Natürliche Zahlen
  - Darstellung zur Basis *b*, Umrechnung
  - Relevante Darstellungen: binär, oktal, dezimal, hexadezimal
- Binäre Arithmetik
- Reelle Zahlen
- Zeichensätze

Zweielementige Boolesche Algebra (1)

Von George Boole 1854 entwickelte Algebra

- Zwei Werte: 0 und 1 (☞ Binäre Zahlenrepräsentation!)
- Drei Operationen: + und * und $\bar{}$
- Vier Axiome/Rechengesetze (nach Huntington, 1904):

1. Kommutativität

   $A + B = B + A$ $\qquad$ $A * B = B * A$

2. Neutrales Element

   $0 + A = A$ $\qquad$ $1 * A = A$

3. Distributivität

   $(A + B) * C = (A * C) + (B * C)$

   $(A * B) + C = (A + C) * (B + C)$

4. Komplementäres Element

   $A + \overline{A} = 1$ $\qquad$ $A * \overline{A} = 0$

Zweielementige Boolesche Algebra (2)

Boolesche Rechenoperationen

– +, Disjunktion, ODER, ∨

∨	**0**	**1**
0	0	1
1	1	1

– *, Konjunktion, UND, ∧

∧	**0**	**1**
0	0	0
1	0	1

– ¯, Negation, Invertierung, NOT, ¬

¬	**0**	**1**
	1	0

Wichtige Sätze (1)

Aus den Axiomen beweisbare Sätze

– Abgeschlossenheit

  Boolesche Operationen liefern nur Boolesche Werte als Ergebnis

– Assoziativität

  $A + (B + C) = (A + B) + C$ $\qquad$ $A * (B * C) = (A * B) * C$

– Idempotenz

  $A + A = A$ $\qquad$ $A * A = A$

– Absorption

  $A + (A * B) = A$ $\qquad$ $A * (A + B) = A$

Wichtige Sätze (2)

Aus den Axiomen beweisbare Sätze

– Doppeltes Komplement

$(\overline{\overline{A}}) = A$

– Komplementäre Werte

$\overline{0} = 1$ $\qquad\qquad$ $\overline{1} = 0$

– Satz von De Morgan

$\overline{(A + B)} = \overline{A} * \overline{B}$ $\qquad\qquad$ $\overline{(A * B)} = \overline{A} + \overline{B}$

Bedeutung der Booleschen Algebra für die Informatik (1)

Computer kennen nur zwei Zustände

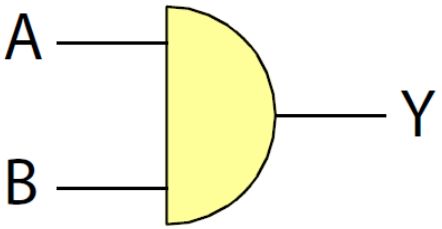
- Informationsgehalt
  - an, aus
  - Strom fließt, fließt nicht
  - wahr, falsch
  - *true*, *false*
  - 1, 0
- Computer-Hardware arbeitet (fast) ausschließlich auf dieser digitalen binären Grundlage
  - Informationsverarbeitung lässt sich vollständig auf Rechenoperationen mit Booleschen Operatoren $\wedge$, $\vee$, $\neg$ zurückführen
  - Informationsspeicherung erfolgt Bit-weise, d.h. in Form einzelner Nullen und Einsen

Bedeutung der Booleschen Algebra für die Informatik (2)

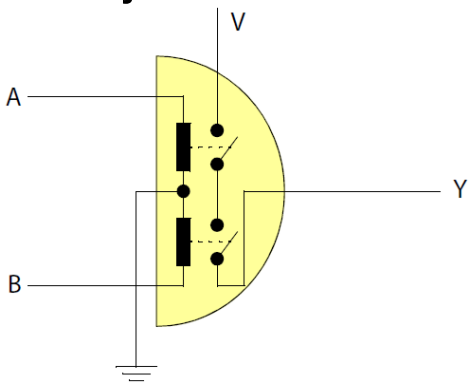
Warum ist das so?

– Speicherung einzelner Bits kann sehr einfach durch Speicherung von elektrischer Ladung realisiert werden, z.B. durch Kondensatoren oder Flip-Flops:
  elektrische Ladung vorhanden → 1
  elektrische Ladung nicht vorhanden → 0

– Konjunktion, Disjunktion und Negation können leicht mittels Schaltern (Transistoren, Relais) realisiert werden, die Ladung (Information) weiterleiten oder nicht. Beispiel */Konjunktion:



$$Y = A \cdot B$$



☞ AND-Gatter

Darstellung positiver Zahlen im Rechner

Zahlenspeicherung in Registern

- Einzelne Flip-Flops speichern eine Ziffer (Bit)
- Mehrere Flip-Flops (Register) speichern eine komplette Zahl

Registerbreite

- 8 Bit = 1 Byte
- Wort = 16 Bit / 32 Bit *(Word)*
- Doppelwort = 32 Bit / 64 Bit *(Double Word)*
- Vierfachwort = 64 Bit / 128 Bit *(Quad Word)*

Wortbreite hängt von Prozessorarchitektur ab

Binäre Addition

Schriftliche Addition / „Schulmethode“

– Verfahren wie beim Dezimalsystem

```
  10011
+  1001
    11
-------
 11100
```

Übertrag

Kontrolle:
19 + 9 = 28

– Übertrag wird auch *Carry* genannt

Feste Registerbreite und Addition, z.B. vier Bits

```
   1011
+  1001
  1 11
-------
  0100
```

Übertrag

Kontrolle:
11 + 9 = 20 = 16 + 4

– Letzter Übertrag gehört zum Ergebnis

– Kann aber nicht mehr dargestellt werden

☞ Überlauf *(Overflow)*

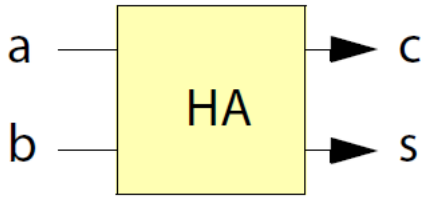
Halbaddierer

Addition in erster (rechter) Spalte

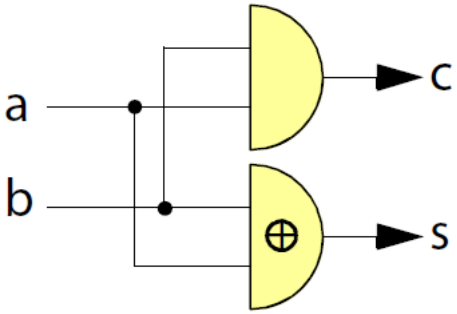
- Zwei Eingänge: erstes Bit von jeder Zahl
- Zwei Ausgänge: erstes Bit des Ergebnisses, *Carry*-Bit
- Wahrheitstabelle

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Blockschaltbild



- Gatterschaltung



$c = a * b$

$s = \overline{a} * b + a * \overline{b} = a \oplus b$ (XOR)

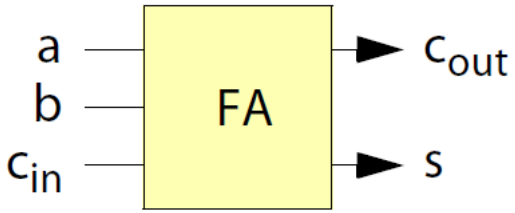
Volladdierer (1)

Addition in anderen Spalte

- Drei Eingänge: je ein Bit der Summanden, und *Carry* von voriger Stelle
- Zwei Ausgänge: Summen-Bit des Ergebnisses, *Carry*-Bit
- Wahrheitstabelle

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

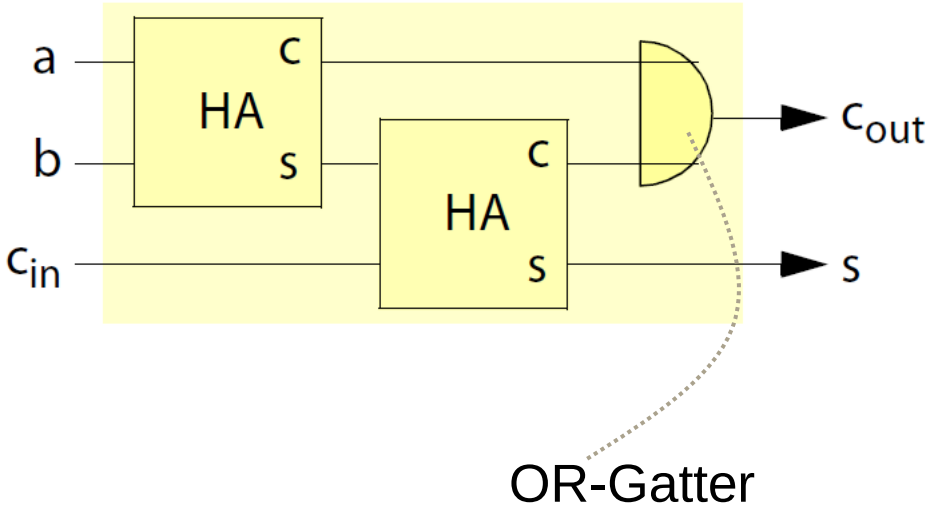
Blockschaltbild



Volladdierer (2)

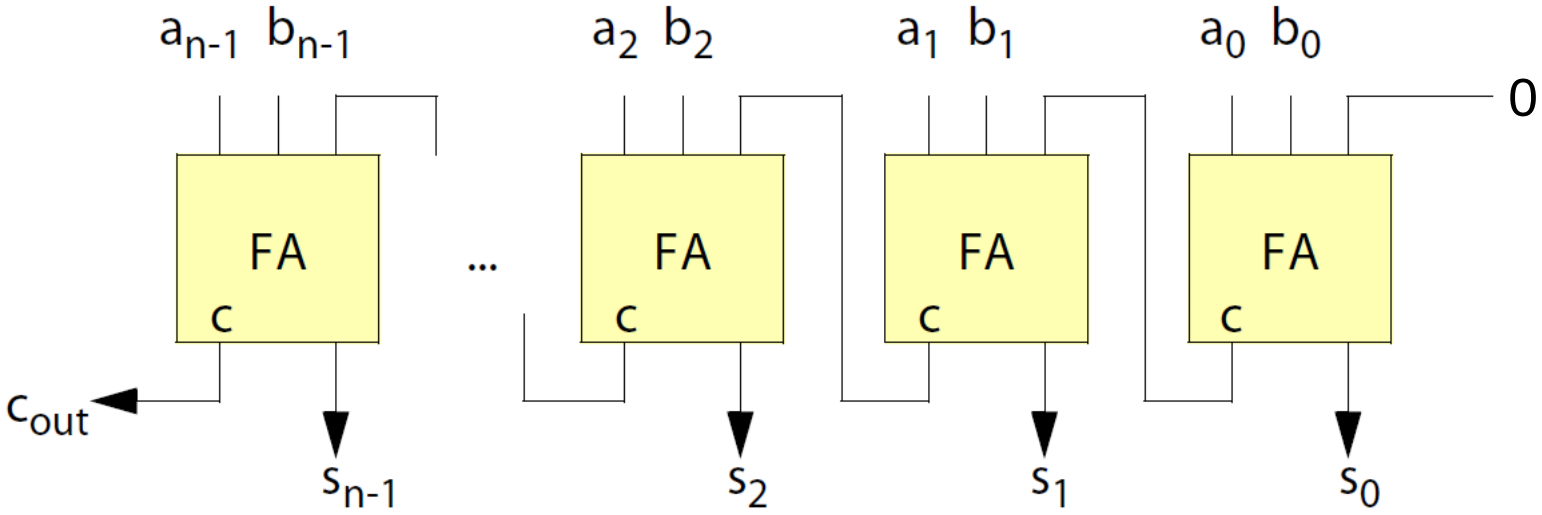
Schaltung

– Aufbau mit Halbaddierern



Paralleles Addierwerk

Schaltung zur Addition *n*-Bit langer Summanden



– *Ripple Carry Adder (RCA)*

Binäre Subtraktion

Subtrahierer kann ähnlich wie Addierer entwickelt werden

- Verwendung von Addierern zur Subtraktion
  - Idee: a – b = a + (-b)

Darstellung negativer Ganzer Zahlen

Vorzeichen und Betrag

- Ein Bit repräsentiert Vorzeichen
- Andere Bits repräsentieren Betrag der Zahl
- Beispiel:

  $01001_2 = 9$

  $11001_2 = -9$

Nachteil

- Vorzeichen muss für Berechnungen explizit ausgewertet werden

Einerkomplement-Darstellung (1)

Berechnung des Einerkomplements einer Zahl *N* bei *n* Ziffern

- $C = 2^n - N - 1$ bei n Ziffern/Bits
- Komplement C entspricht dem Wert $-N$

Darstellung positiver ganzer Zahlen

- Höchstwertiges Bit $z_{n-1} = 0$
- Andere Bits unbeschränkt
- Wert: $(z_{n-1}, ..., z_1, z_0)_2 = \sum_i z_i * 2^i$

Darstellung negativer ganzer Zahlen

- Höchstwertiges Bit $z_{n-1} = 1$
- Andere Bits unbeschränkt
- Wert: $(z_{n-1}, ..., z_1, z_0)_2 = -2^n + 1 + \sum_i z_i * 2^i$

Einerkomplement-Darstellung (2)

Beispiel: Darstellungslänge $n = 4$

- $1000_2 = -2^4 + 1 + 8 = -7$ ☞ kleinste negative Zahl ($-2^{n-1}+1$)
  $= \overline{+7} = \overline{0111_2} = 1000_2$
- $0111_2 = 7$ ☞ größte positive Zahl ($2^{n-1}-1$)
- $1111_2 = -2^4 + 1 + 15 = 0$ ☞ größte negative Zahl
  $= \overline{+0} = \overline{0000_2} = 1111_2$
- $0000_2 = 0$ ☞ kleinste positive Zahl

Nachteile

- Null hat zwei Darstellungen, explizite Vorzeichenbehandlung

Vorteil

- Einfache Umwandlung von positiver zu negativer Zahl und umgekehrt
  - Jede Ziffer wird invertiert: $z_i' = 2 - 1 - z_i$
  - Beispiel: aus 1000_2 wird 0111_2 (aus -7 wird +7)

Einerkomplement-Darstellung (3)

Einerkomplement heute kaum mehr im Einsatz

– Doppelte Darstellung der Null
– Kompliziertere Hardware zur Addition/Subtraktion

„Daseinsberechtigung“ des Einerkomplements

– Zur Motivation und Überleitung zum Zweierkomplement

Zweierkomplement-Darstellung (1)

Berechnung des Zweierkomplements einer Zahl *N* bei *n* Ziffern

- $C = 2^n - N$ bei n Ziffern/Bits
- Komplement C entspricht dem Wert $-N$

Darstellung positiver ganzer Zahlen

- Höchstwertiges Bit $z_{n-1} = 0$
- Andere Bits unbeschränkt
- Wert: $(z_{n-1}, ..., z_1, z_0)_2 = \sum_i z_i * 2^i$

Darstellung negativer ganzer Zahlen

- Höchstwertiges Bit $z_{n-1} = 1$
- Andere Bits unbeschränkt
- Wert: $(z_{n-1}, ..., z_1, z_0)_2 = -2^n + \sum_i z_i * 2^i$

Zweierkomplement-Darstellung (2)

Beispiel: Darstellungslänge $n = 4$

- $1000_2 = -2^4 + 8 = -8$ ☞ kleinste negative Zahl (-2^{n-1})
- $0111_2 = 7$ ☞ größte positive Zahl ($2^{n-1}-1$)
- $1111_2 = -2^4 + 15 = -1$ ☞ größte negative Zahl
- $0000_2 = 0$ ☞ kleinste positive Zahl

Vorteil des Zweierkomplements

- Eindeutige Darstellung der Null (0000_2 bei Länge $n = 4$)
- Einfache Umwandlung von positiver zu negativer Zahl und umgekehrt
  - Jede Ziffer wird invertiert: $z_i' = 2 - 1 - z_i$
  - Anschließend 1 auf niederwertigste Stelle addieren (Zweierkomplement ist um eins größer als Einerkomplement)
  - Beispiel: aus 1001_2 wird 0110_2 und dann 0111_2 (aus -7 wird +7)

Zweierkomplement-Darstellung (3)

Nachteil des Zweierkomplements

- Für größte positive Zahl ist das Zweierkomplement nicht mehr darstellbar
  - 8 wird zu 1000_2 = -8
  - 8 bereits außerhalb des Darstellungsbereichs (Überlauf)

Addition

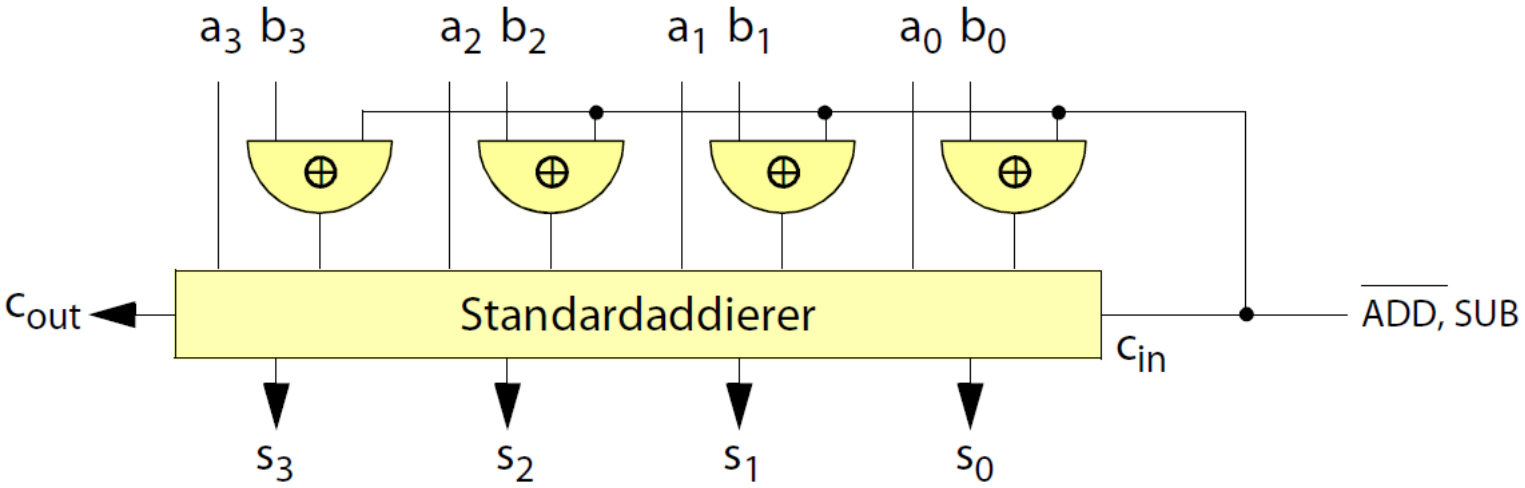
- Einsatz von Standardaddierern für Zahlen im Zweierkomplement

Subtraktion

- Vorherige Komplementbildung eines Summanden erfordert
  - Invertierung der Ziffern
  - Addition von 1
    kann durch gesetzten Carry-Eingang erzielt werden

Subtraktion im Zweierkomplement

Addier- und Subtrahierwerk

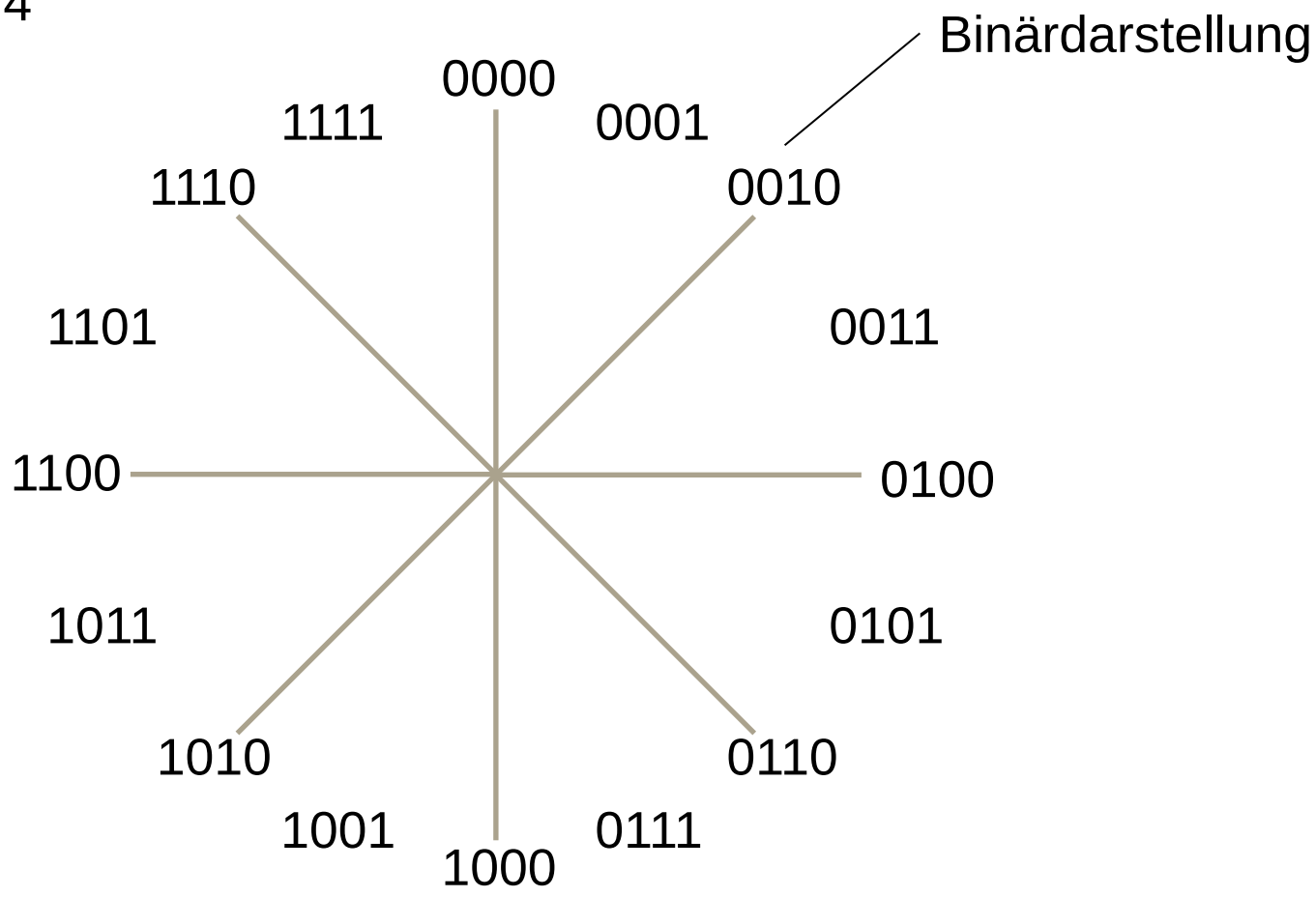


- Beim Subtrahieren
  - Invertieren der b-Eingänge durch XOR-Gatter
  - Addieren von 1 durch gesetztes Carry-in
- Überlauferkennung: $c_{out} \neq c_{in}$

Zahlenraum der Zweierkomplement-Darstellung (1)

Zahlenraum für *n*-stellige Register

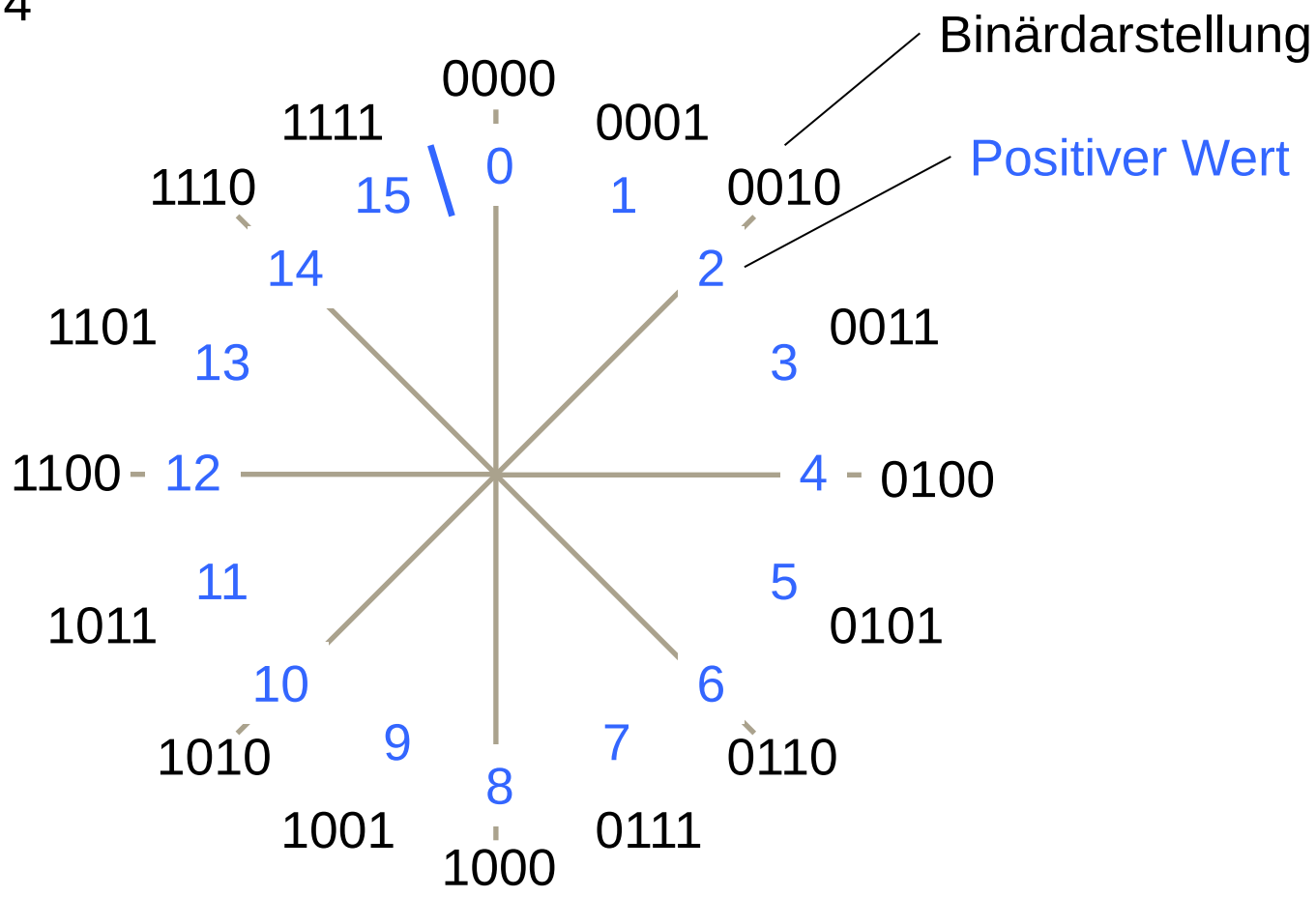
– Beispiel: $n = 4$



Zahlenraum der Zweierkomplement-Darstellung (2)

Zahlenraum für *n*-stellige Register

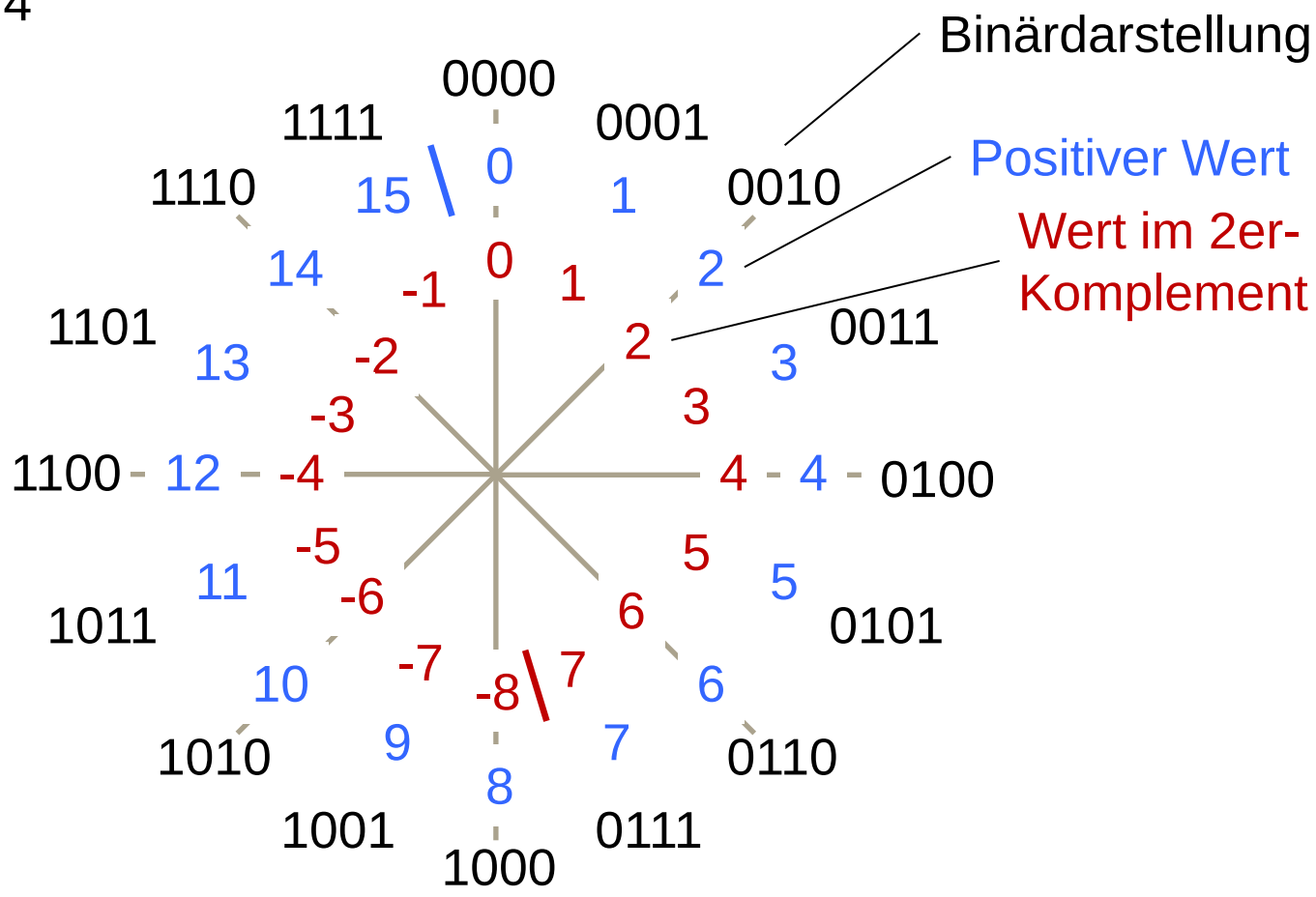
– Beispiel: $n = 4$



Zahlenraum der Zweierkomplement-Darstellung (3)

Zahlenraum für *n*-stellige Register

– Beispiel: $n = 4$



Binäre Multiplikation

Schriftliche Multiplikation / „Schulmethode“ auf positiven Binärzahlen

```
      0011 * 1010
      ---------
      0011        1
     0000         0
    0011          1
+    0000         0
  = 0011110
```

Ver-schieben

Addieren

Kontrolle:
3 * 10 = 30
$30 = 11110_2$

Übertragung auf einen Computer

– Realisierung der Multiplikation durch Verschiebe-Operationen (Schieberegister) und einen Addierer

Multiplikation im Zweierkomplement (1)

_n_ Bit breites Ergebnis bei Multiplikation _n_ Bit breiter Zahlen auch für Zweierkomplement korrekt

– Beispiel: -2 * 3 = -6 (für $n = 4$)

```
      1110 * 0011
      ----------
      0000        0
       0000       0
        1110      1
+        1110     1
= 00101010
```

$1010_2 = -6$

Problem: Überlauf – Ergebnis passt meist nicht in _n_ Bits

– $2n$ Bit breites Ergebnis ist nicht korrekt

$00101010_2 = 42$

Multiplikation im Zweierkomplement (2)

Alternative A: Erweiterung der Faktoren auf 2*n* Bits Breite

– Vorzeichenerweiterung
  – Z.B. aus 1110 wird 11111110, aus 0011 wird 00000011

Nachteil

– 2*n* Bit breiter Addierer notwendig
– 2*n* anstatt von *n* Runden

Multiplikation im Zweierkomplement (3)

Alternative B: Addition eines Korrektursummanden

- $a * (-b) = a * (2^n - b) = 2^n * a - (a * b)$
- $(-a) * b = (2^n - a) * b = 2^n * b - (a * b)$
- $(-a) * (-b) = (2^n - a) * (2^n - b) = 2^{2n} - 2^n * a - 2^n * b + a * b$

→ Gleichheit gilt wg. Definition 2er-Komplement (☞ Folie 26)

Ziel: $2n$ Bit breites 2er-Komplement des Produkts a * b: $2^{2n} - (a * b)$

- Korrektursummand S ist gleich der Differenz zwischen gewünschter $2n$ Bit breiter Zahl und dem bisherigen (falschen) Multiplikationsergebnis
- Fall a * (-b): $S = 2^{2n} - (a * b) - (a * -b)$

  $= 2^{2n} - (a * b) - (2^n * a - a * b) = 2^{2n} - 2^n * a = 2^n * (2^n - a)$
- Fall -a * b: $S = 2^{2n} - 2^n * b = 2^n * (2^n - b)$
- Fall -a * -b: $S = 2^n * a + 2^n * b$

Nachteil: Hoher Zusatzaufwand

Multiplikation im Zweierkomplement (4)

Alternative C: Getrennte Behandlung des Vorzeichens

- Umwandlung der Faktoren in positive Zahlen
- Berechnung des Ergebnisvorzeichens
- Anpassen des Ergebnisses

Nachteil: Hoher Zusatzaufwand

Multiplikation im Zweierkomplement (5)

Alternative D: Verfahren nach Booth

- Idee: a * 0111 = a * 1000 – a * 0001
  - Gilt auch für skalierte Bitfolge,
    z.B. a * 011100 = a * 100000 – a * 000100
  - Komplette Folge von 1-Bits lässt sich durch genau *eine* Addition (am „linken Rand“ der 1-Folge) und *eine* Subtraktion („rechter Rand“) multiplizieren
- Booth-Algorithmus
  - Betrachte „Fenster“ von 2 Bits (b_i, b_{i-1}), das über Faktor b von rechts nach links geschoben wird (b_{-1} sei als 0 definiert)
  - $(b_i, b_{i-1})_2 = 01_2$ Addiere a * 2^i (linker Rand einer 1-Folge)
  - $(b_i, b_{i-1})_2 = 10_2$ Subtrahiere a * 2^i (rechter Rand)
  - $(b_i, b_{i-1})_2 = 11_2$ oder 00_2 Tue nichts

Multiplikation im Zweierkomplement (6)

Alternative D: Verfahren nach Booth

- Subtraktion durch Addition des Zweierkomplements
- Gültige Ergebnisse auch für negative Zahlen (☞ Zweierkomplement)
- *n* Bits breiter Addierer ist ausreichend durch geschicktes Schieben
  - Schiebeoperationen mit Vorzeichenpropagierung (Vorzeichen wird verdoppelt)

Multiplikation im Zweierkomplement (7)

Alternative D: Verfahren nach Booth

– Beispiel: -2 * 3 = -6 (für n = 4)

Fenster

a * b: **1110 * 0011**

$2n$+1 Bits breite Hilfsvariable r: **0000 0011 0**

Bits r_4, ..., r_1 mit b initialisiert, Rest 0

1. Fenster „10“: Subtrahiere a: **-1110**

   0010 0011 0

   Vorzeichenbehaftetes Schieben: **0001 0001 1**

2. Fenster „11“: nur Schieben: **0000 1000 1**

3. Fenster „01“: Addiere a: **+1110**

   1110 1000 1

   Vorzeichenbehaftetes Schieben: **1111 0100 0**

4. Fenster „00“: nur Schieben: **1111 1010 0**

Ergebnis: 11111010_2 = -6

Multiplikation im Zweierkomplement (8)

Alternative D: Verfahren nach Booth

– Beispiel: 3 * -2 = -6 (für n = 4)

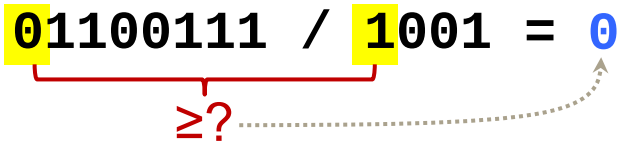
a * b:	**0011 * 1110**
$2n$+1 Bits breite Hilfsvariable r:	**0000 1110 0**
1. Fenster „00“: nur Schieben:	**0000 0111 0**
2. Fenster „10“: Subtrahiere a:	**-0011**
	1101 0111 0
Vorzeichenbehaftetes Schieben:	**1110 1011 1**
3. Fenster „11“: nur Schieben:	**1111 0101 1**
4. Fenster „11“: nur Schieben:	**1111 1010 1**

Ergebnis: 11111010_2 = -6

Binäre Division (1)

Schriftliche Division / „Schulmethode“

– Beispiel: 103 / 9 = ?



Binäre Division (2)

Schriftliche Division / „Schulmethode“

– Beispiel: 103 / 9 = ?

```
01100111 / 1001 = 0
0000
```

*

Binäre Division (3)

Schriftliche Division / „Schulmethode“

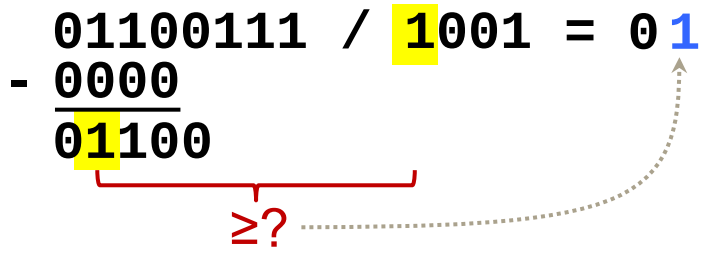
– Beispiel: 103 / 9 = ?

```
  01100111 / 1001 = 0
- 0000
  01100
```

Binäre Division (4)

Schriftliche Division / „Schulmethode“

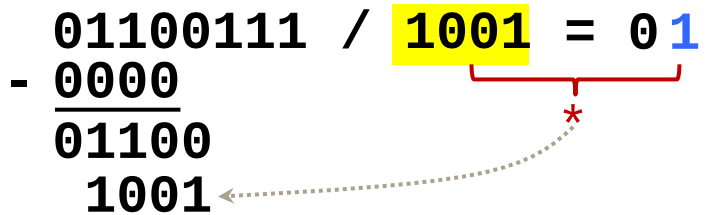
– Beispiel: 103 / 9 = ?



Binäre Division (5)

Schriftliche Division / „Schulmethode“

– Beispiel: 103 / 9 = ?



Binäre Division (6)

Schriftliche Division / „Schulmethode“

– Beispiel: 103 / 9 = ?

```
  01100111 / 1001 = 01
- 0000
  01100
-  1001
   00111
```

Binäre Division (7)

Schriftliche Division / „Schulmethode“

– Beispiel: 103 / 9 = ?

```
  01100111 / 1001 = 01011
- 0000
  01100
-  1001
   00111
-   0000
    01111
-    1001
     01101
-     1001
      0100
```

Quotient

Divisor

Dividend

Rest

Kontrolle:
103 / 9 = 11 Rest 4

Binäre Division (8)

Mathematisch

– Dividend = Quotient * Divisor + Rest

oder

Dividend / Divisor = Quotient + Rest / Divisor

– Häufig verlangt: Rest hat gleiches Vorzeichen wie Dividend
– Überlauf möglich, wenn Quotient nicht so breit wie Dividend

– Realisierung über Addierer/Subtrahierer und Schiebeoperationen pro einzelnem Schritt

Roter Faden

2. Zahlendarstellungen und Rechnerarithmetik

- Natürliche Zahlen
- Binäre Arithmetik
  - Boolesche Algebra
  - Addition: Halbaddierer, Volladdierer
  - Subtraktion: Zweierkomplement-Darstellung
  - Multiplikation: Booth-Algorithmus
  - Division: „Schulmethode“
- Reelle Zahlen
- Zeichensätze

Festkomma-Darstellung

Feste Kommaposition bei der Darstellung von Zahlen

- Beispiel: $n = 4$, Komma an Position $k = 2$
  - Registerinhalt 0110 bedeutet $01{,}10_2$ bedeutet $1{,}5_{10}$
- Allgemeine Wertberechnung (für positive Zahlen):

$$(z_{n-k-1}, \ldots, z_1, z_0, z_{-1}, \ldots, z_{-k})_2 = \sum_i z_i * 2^i$$

- Negative Zahlen analog

Rechenoperationen

- Addition und Subtraktion: unverändert
- Multiplikation: Ergebnis hat $2k$ Nachkommastellen

  ☞ Skalieren / Abschneiden auf k Nachkommastellen
- Division: Einfügen des Ergebniskommas, sobald erste Nachkommastelle des Dividenden berührt wird

Gleitkomma-Darstellung (1)

Ziel: Darstellung großer *und kleiner* Zahlen mit gleichem Verfahren

- Datentyp **`real`** oder **`float`** aus gängigen Programmiersprachen
- **Vorsicht:** Gleitkommazahlen entsprechen nicht reellen Zahlen im mathematischen Sinne, sondern bloß einer Annäherung!

Idee

- Darstellung einer Anzahl von Ziffern *(Mantisse)* **plus**
- Darstellung der Position des Kommas (Gleitkomma)
- Beispiele:
  - 12.345 / k = 2: 123,45
  - 12.345 / k = 5: 0,123.45
  - 12.345 / k = -4: 123.450.000,0
- Beispiel: Wissenschaftliche Notation des Taschenrechners
  - $1{,}234.5 * 10^4$ entspricht 12.345,0
  - *Exponent* zur Basis 10 gibt Position des Kommas an

Gleitkomma-Darstellung (2)

Allgemein

- Zahl x wird dargestellt als: $x = m * b^e$
  (Mantisse m multipliziert mit Exponent e zur Basis b)
- e wird auch *Charakteristik* genannt

Normalisierung

- Zahl $x \neq 0$ heißt normalisiert, wenn gilt: $1 \leq m < b$
- Beispiel für $b = 10$
  - 12.345 wird dargestellt als $1{,}2345 * 10^4$
  - Wert der Mantisse liegt zwischen 1 und 10
- Beispiel für $b = 2$
  - 3,625 wird dargestellt als $1{,}1101_2 * 2^1$
  - Wert der Mantisse liegt zwischen 1 und 2

Binäre Darstellung von Gleitkommazahlen

Freiheitsgrade bei der Darstellung

- Gesamtlänge der Darstellung
- Länge der Exponentendarstellung (Länge der Mantissendarstellung)
- Darstellung der Mantisse
  (Einer- oder Zweierkomplement, oder Vorzeichen und Betrag)
- Darstellung des Exponenten
  (Einer- oder Zweierkomplement, Vorzeichen und Betrag, oder *Biased Exponent*)

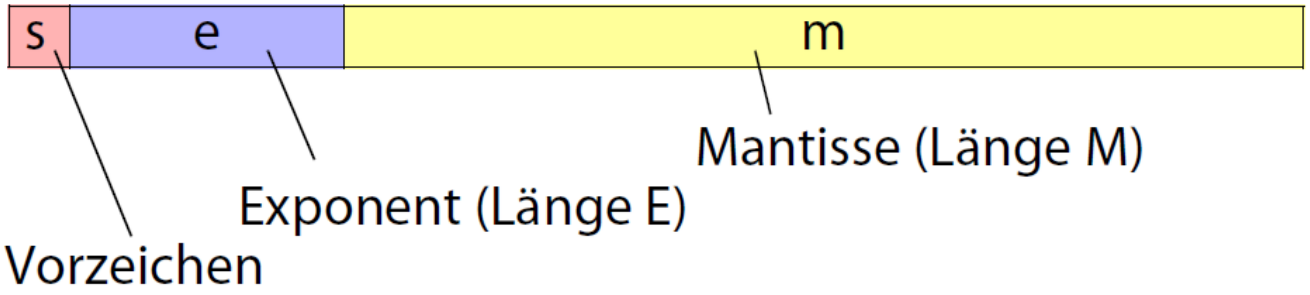
Biased Exponent

- Darstellung des Exponenten ist immer positiv und um eine Konstante *(Bias)* größer als der tatsächliche Wert
- Beispiel: *Bias B* = 63, Exponent e = -8 Darstellung e_{Darst} = 55
- Vorteil: Durchgängig positiver Zahlenraum für die Charakteristik

IEEE 754 Gleitkomma-Darstellung (1)

Standard zur Vereinheitlichung der unterschiedlichen Darstellungen

Aufbau einer IEEE 754 Gleitkommazahl



- Allgemeine Wertberechnung: $x = (-1)^s * 1,m * 2^{e-B}$
- Erste Ziffer (immer 1) wird nicht in Mantisse gespeichert (sog. verdecktes Bit, *hidden bit*)
- Bias B hängt von der Länge der Exponentendarstellung E ab: $B = 2^{E-1} - 1$
- Gültige Charakteristiken: $0 < e < 2^E - 1$
  (Werte 0 und $2^E - 1$ sind reserviert)

IEEE 754 Gleitkomma-Darstellung (2)

Spezielle Werte

- Null / *Zero*
  - 0 ist nicht als normalisierte Zahl darstellbar, daher gesonderte Behandlung
  - Vorzeichen s, $e = 0$, $m = 0$ (☞ es existiert positive und negative Null)
- Unendlich / *Infinity*
  - Symbolische Darstellung für unendlich große / kleine Zahl; sinnvoll bspw. bei Überläufen, anstatt der Rückgabe der größten darstellbaren Zahl.
  - Beispiel: $\sqrt{x^2 + y^2}$ mit Überlauf bei x^2; Ergebnis ∞ statt normaler Zahl
  - Vorzeichen s, $e = 2^E - 1$, $m = 0$ (positiv und negativ unendlich)

IEEE 754 Gleitkomma-Darstellung (3)

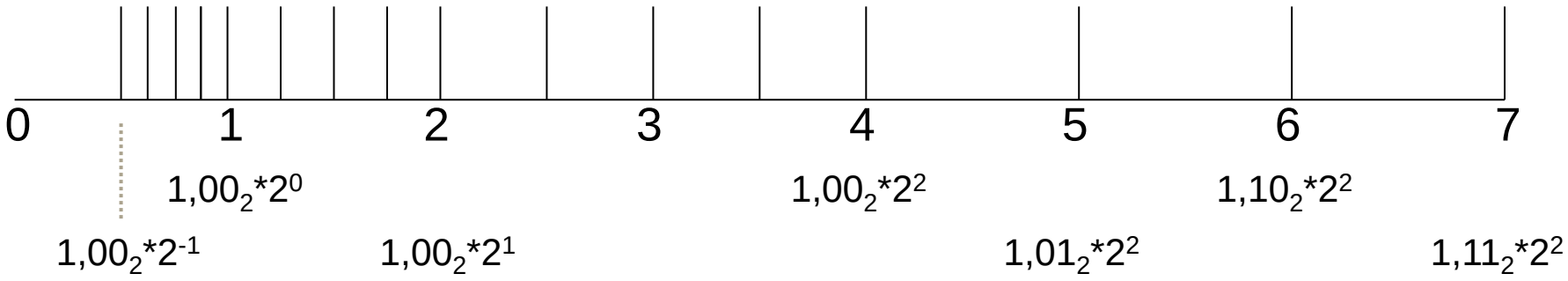
Spezielle Werte

- NaN / *Not a number*
  - Falls Rechenoperation für bestimmte Argumente nicht definiert ist, wird NaN zurückgegeben.
  - Beispiel: $\sqrt{x}$ für $x < 0$ liefert NaN
  - Vorzeichen s, $e = 2^E - 1$, $m \neq 0$

IEEE 754 Gleitkomma-Darstellung (4)

Spezielle Werte

- Nicht-normalisierte Zahlen (kleiner als kleinste normalisierte Zahl)
  - Problem: Abstand zwischen darstellbaren normalisierten Zahlen in der Nähe der Null (verhältnismäßig) groß. Beispiel für $e \in [-1, ..., 2]$



  - Nicht-normalisierte Zahlen entstehen rund um die Null, indem das *hidden bit* der Mantisse als 0 anstatt von 1 angenommen wird
  - Vorzeichen s, $e = 0$, $m \neq 0$
  - Wertberechnung: $x = (-1)^s * \underline{\mathbf{0}},m * 2^{1-B}$

IEEE 754 Gleitkomma-Darstellung (5)

Formatdefinitionen

	Single Precision	***Double Precision***	***Quad Precision***
Gesamtlänge *(N)*	32 Bit	64 Bit	128 Bit
Vorzeichen	1 Bit	1 Bit	1 Bit
Mantisse *(M)*	23 Bit	52 Bit	112 Bit
Exponent *(E)*	8 Bit	11 Bit	15 Bit
Bias *(B)*	127	1023	16383
$\lvert x_{min} \rvert$ (norm.)	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$	$2^{-16382} \approx 10^{-4932}$
$\lvert x_{min} \rvert$ (denorm.)	$2^{-149} \approx 10^{-45}$	$2^{-1074} \approx 10^{-324}$	$2^{-16492} \approx 10^{-4965}$
$\lvert x_{max} \rvert$	$(2 - 2^{-23}) * 2^{127} \approx 10^{38}$	$(2 - 2^{-52}) * 2^{1023} \approx 10^{308}$	$(2 - 2^{-112}) * 2^{16383} \approx 10^{4932}$

– Zusätzliches Format: *Extended Precision* zwischen *Double* und *Quad* (herstellerabhängig definierbar)

Gleitkomma-Rechenoperationen

Beispiel IEEE 754 Darstellung

Addition / Subtraktion

- Denormalisiere Zahl mit kleinerem Exponent
  - D.h. Exponenten auf gleichen Wert bringen
- Addiere oder subtrahiere Mantissen
- Normalisiere Mantisse
- Berechne Vorzeichen des Ergebnisses

Multiplikation / Division

- Multipliziere / dividiere Mantissen
- Addiere / subtrahiere Exponenten
- Normalisiere Mantisse
- Berechne Vorzeichen des Ergebnisses

Roter Faden

2. Zahlendarstellungen und Rechnerarithmetik

- Natürliche Zahlen
- Binäre Arithmetik
- Reelle Zahlen
- Zeichensätze

Zeichensätze

Repräsentation von Texten

- Naiver Ansatz:
  Codierung A $\rightsquigarrow$ 0, B $\rightsquigarrow$ 1, ...
  und dann Binärcodierung
- Probleme:
  - Welche Zeichen sollen codiert werden?
  - Wie kann man Daten/Texte mit anderen austauschen?

Lösung

☞ Standardisierte Zeichensätze

ASCII (1)

American Standard Code for Information Interchange

- Verabschiedet 1963 von der *American Standards Organization*
- 7-Bit Code
- Für die USA gedacht
- Codiert Zeichen und Steuercodes zur Kontrolle von Geräten; z.B. CR *(carriage return)* für Wagenrücklauf bei Druckern; BEL für Glocke

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

[de.wikipedia.org]

ISO-8859-1 (1)

Motivation

– Problem: Viele wichtige Zeichen fehlen bei ASCII
– Ansatz: „Längere Codierung“, Erweiterung des Zeichensatzes

ISO-8859-1 / ISO Latin 1

– ISO = *International Organization for Standardization*
– 8-Bit Code
– Enthält viele Sonderzeichen für westeuropäische Sprachen (z.B. Umlaute etc.)

ISO-8859-1 (2)

Zeichentabelle

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	*nicht belegt*															
1...																
2...	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8...	*nicht belegt*															
9...																
A...	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
B...	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C...	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D...	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E...	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F...	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

ASCII

[de.wikipedia.org]

Problem

– Einige wichtige Zeichen (französische Sonderzeichen, €-Symbol) fehlen nach wie vor

Unicode (1)

Fakten

- Verwaltet vom Unicode-Konsortium (**`http://www.unicode.org`**)
- Unterstützt verschiedene Codierungsformate *(Unicode Transformation Format)*: UTF-8, UTF-16, UTF-32 mit 8, 16, 32 Bits
- Längere Unicode-Formate ergänzen kürzere Formate
- Unicode vereinbart auch weitere Informationen (z.B. Schreibrichtung)
- Ziel: Codierung *aller* in Gebrauch befindlicher Schriftsysteme und Zeichen

☞ Unicode wird kontinuierlich um neue Zeichen ergänzt

Unicode (2)

Gliederung

- 17 *Planes* à je 65.536 Zeichen
  6 *Planes* derzeit genutzt, restliche sind für spätere Nutzung vorgesehen
- *Basic Multilingual Plane (BMP, 0):* grundlegender mehrsprachiger Codebereich, enthält aktuell gebräuchliche Schriftsysteme, Satzzeichen und Symbole
- *Supplementary Multilingual Plane (SMP, 1):* historische Schriftsysteme, weniger gebräuchliche Zeichen (z.B. Domino- und Mahjonggsteine)
- *Supplementary Ideographic Plane (SIP, 2):* ergänzender ideographischer Bereich für selten benutzte fernöstliche CJK-Schriftzeichen
  (CJK = China, Japan, Korea)

[de.wikipedia.org]

Zusammenfassung (1)

Natürliche Zahlen

- Positionale Darstellung: Ziffern, Position der Ziffern als Gewichtung
- Zahlen darstellbar mit / umrechenbar in jeder beliebigen Basis *b*
- Wichtig: *b* = 2 (binär), 8 (oktal), 10 (dezimal), 16 (hexadezimal)

Binäre Arithmetik

- Boolesche Algebra: Grundlage für Computer-Hardware; Werte 0 und 1; Rechnen über Operationen Kunjunktion, Disjunktion und Negation
- Addition: Halbaddierer addiert zwei Bits, produziert Summe und *Carry*; Volladdierer addiert zwei Bits und *Carry*; Addierwerk aus Volladdierern
- Subtraktion: 1er-Komplement für ganze Zahlen obsolet; 2er-Komplement
- Multiplikation: Booth-Algorithmus braucht nur 1 Addition und 1 Subtraktion zum Multiplizieren mit einer beliebig langen 1-Folge
- Division: Schulmethode beruht auf Additionen und Schiebeoperationen

Zusammenfassung (2)

Rationale Zahlen

– Festkommazahlen: Komma stets an gleicher Bit-Position; Grundrechenarten verhältnismäßig leicht
– Gleitkommazahlen: Mantisse und Exponent / Charakteristik; normalisierte Mantisse; *biased exponent*
– IEEE 754: normierte Zahlendarstellung; *hidden bit*; spezielle Werte: Null, unendlich, NaN, nicht-normalisierte Zahlen; Grundrechenarten: gesonderte Behandlung von Exponent und Mantisse, abschließende Normalisierung

Zeichensätze

– ASCII, ISO Latin 1, Unicode