

# 64-040 Modul IP7: Rechnerstrukturen

## 2. Informationsbegriff und Repräsentation

Norman Hendrich & Jianwei Zhang

Universität Hamburg  
MIN Fakultät, Department Informatik  
Vogt-Kölln-Str. 30, D-22527 Hamburg  
{hendrich,zhang}@informatik.uni-hamburg.de

WS 2010/2011



# Inhalt

## Information

Definition und Begriff

## Zahldarstellung

Konzept der Zahl

Stellenwertsystem

Umrechnung zwischen verschiedenen Basen

Zahlenbereich und Präfixe

Festkommazahlen

Darstellung negativer Zahlen

Gleitkomma und IEEE 754

Literatur



# Information

- ▶ **Information** := abstrakter Gehalt einer Aussage
- ▶ Die Aussage selbst, mit der die Information dargestellt bzw. übertragen wird, ist eine **Repräsentation** der Information
- ▶ Das Ermitteln der Information aus einer Repräsentation heisst **Interpretation**
- ▶ Das Verbinden einer Information mit ihrer Bedeutung in der realen Welt heisst **Verstehen**



## Repräsentation (Beispiele)

Beispiel: Mit der Information „25“ sei die abstrakte Zahl gemeint, die sich aber nur durch eine Repräsentation angeben lässt:

- ▶ Text deutsch:                    fünfundzwanzig
- ▶ Text englisch:                    twentyfive
- ... ..
- ▶ Zahl römisch:                    XXV
- ▶ Zahl dezimal:                    25
- ▶ Zahl binär:                    11001
- ▶ Zahl Dreiersystem:            221
- ... ..
- ▶ Morse-Code:                    ..--- .....



## Information vs. Interpretation

- ▶ Wo auch immer Repräsentationen auftreten, meinen wir eigentlich die Information, z.B.:

$$5 \cdot (2 + 3) = 25$$

- ▶ Die Information selbst kann man überhaupt nicht notieren (!)
- ▶ Es muss immer Absprachen geben über die verwendete Repräsentation. Im obigen Beispiel ist implizit die Dezimaldarstellung gemeint, man muss also die Dezimalziffern und das Stellenwertsystem kennen.
- ▶ Repräsentation ist häufig mehrstufig, z.B.
 

Zahl:	Dezimalzahl:	347
Ziffer:	4-bit binär:	0011 0100 0111
Bit:	elektrische Spannung:	0.1V 0.1V 3.3V 3.3V ...

## Repräsentation vs. Ebenen

In jeder (Abstraktions-) Ebene gibt es beliebig viele Alternativen der Repräsentation

- ▶ Auswahl der jeweils effizientesten Repräsentation
- ▶ unterschiedliche Repräsentationen je nach Ebene

Beispiel: Repräsentation der Zahl  $\pi = 3.1415\dots$  im

- |                 |                                     |
|-----------------|-------------------------------------|
| ▶ x86 Prozessor | 80-bit Binärdaten, Spannungen       |
| ▶ Hauptspeicher | 64-bit Binärdaten, Spannungen       |
| ▶ Festplatte    | kodierte Zahl, magnetische Bereiche |
| ▶ CD-ROM        | kodierte Zahl, Land/Pits-Bereiche   |
| ▶ Papier        | Text, „3.14159265...“               |
| ▶ ...           |                                     |

## Information vs. Nachricht

N1: Er besucht General Motors

N2: Unwetter am Alpenostrand

N3: Sie nimmt ihren Hut

- ▶ Alle Aussagen sind aber doppel/mehrdeutig:

N1: Firma? Militär?

N2: Alpen-Ostrand? Alpeno-Strand?

N3: tatsächlich oder im übertragenen Sinn?

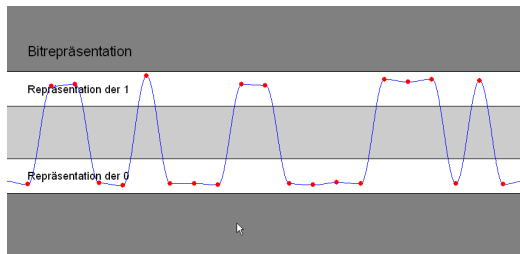
**Interpretation:** Es handelt sich um drei **Nachrichten**, die jeweils zwei verschiedene **Informationen** enthalten

## Information vs. Repäsentation

- ▶ **Information:** Wissen um oder Kenntnis über Sachverhalte und Vorgänge (Der Begriff wird nicht informationstheoretisch abgestützt, sondern an umgangssprachlicher Bedeutung orientiert).
- ▶ **Nachricht:** Zeichen oder Funktionen, die Informationen zum Zweck der Weitergabe aufgrund bekannter oder unterstellter Abmachungen darstellen (DIN 44 300).
- ▶ Beispiel für eine Nachricht: Temperaturangabe in Grad Celsius oder Fahrenheit.
- ▶ Die Nachricht ist also eine Darstellung von Informationen und nicht der Übermittlungsvorgang.

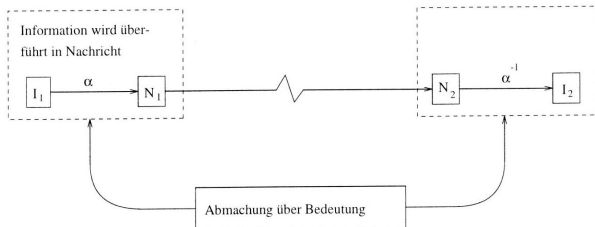


# Beispiel: Binärwerte in 5 V-CMOS-Technologie



- ▶ Spannungsverlauf des Signals ist kontinuierlich
- ▶ Abtastung zu bestimmten Zeitpunkten
- ▶ Quantisierung über abgegrenzte Wertebereiche:
  - ▶  $0.0\text{ V} \leq a(t) \leq 1.2\text{ V}$ : Interpretation als 0
  - ▶  $3.3\text{ V} \leq a(t) \leq 5.0\text{ V}$ : Interpretation als 1
  - ▶ außerhalb und innerhalb: ungültige Werte

# Modell der Informationsübertragung

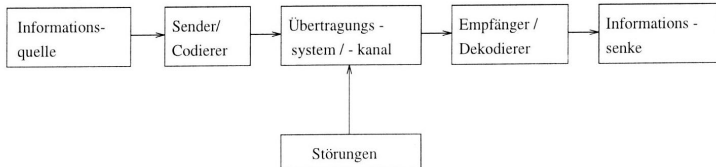


## Beschreibung der **Informationsübermittlung**:

- ▶ die Nachricht  $N_1$  entsteht durch Abbildung  $\alpha$  aus der Information  $I_1$
- ▶ Übertragung der Nachricht an den Zielort
- ▶ Umkehrabbildung  $\alpha^{-1}$  aus der Nachricht  $N_2$  liefert die Information  $I_2$



# Nachrichtentechnisches Modell der Informationsübertragung

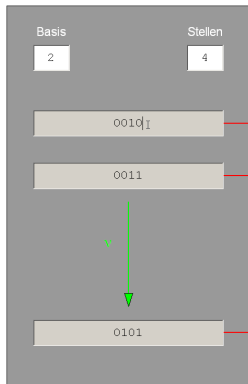


## Beispiele für **Störungen**:

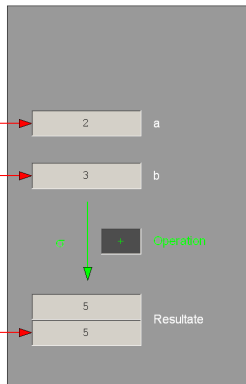
- ▶ Bitfehler beim Speichern
- ▶ Störungen beim Funkverkehr
- ▶ Schmutz oder Kratzer auf einer CD/DVD
- ▶ USW.

# Demo: Information vs. Repräsentation

## Repräsentation



## Information



Interpretation

(Klaus von der Heide, Interaktives Skript T1, infopres)

# Informationstreu

ergibt  $\alpha$  gefolgt von  $\sigma$  dasselbe wie  $\nu$  gefolgt von  $\alpha'$ , dann heisst  $\nu$  **informationstreu**.

- ▶ mit  $\alpha'$  als der Interpretation des Resultats der Operation  $\nu$
- ▶ häufig sind  $\alpha$  und  $\alpha'$  gleich, aber nicht immer
- ▶  $\sigma$  injektiv: **Umschlüsselung**
- ▶  $\nu$  injektiv: **Umcodierung**
- ▶  $\sigma$  innere Verknüpfung der Menge  $\mathcal{J}$  und  $\nu$  innere Verknüpfung der Menge  $\mathcal{R}$ : dann ist  $\alpha$  ein Homomorphismus der algebraischen Strukturen  $(\mathcal{J}, \sigma)$  und  $(\mathcal{R}, \nu)$ .
- ▶  $\sigma$  bijektiv: Isomorphismus

## Informationstreue

- ▶ Welche mathematischen Eigenschaften gelten bei der Informationsverarbeitung, in der gewählten Repräsentation?
- ▶ Beispiel: gilt  $x^2 \geq 0$ ?
  - ▶ float: ja
  - ▶ signed integer: nein
- ▶ Gilt  $(x + y) + z = x + (y + z)$ ?
  - ▶ integer: ja
  - ▶ float: nein  
 $1.0E20 + (-1.0E20 + 3.14) = 0$
- ▶ Details: später



# Beschreibung von Information durch Zeichen

**Zeichen** (engl: *character*): Element  $z$  aus einer zur Darstellung von Information vereinbarten, einer Abmachung unterliegenden, endlichen Menge  $Z$  von Elementen.

Die Menge heisst **Zeichensatz** oder **Zeichenvorrat** (engl. *character set*).

## Beispiel:

- ▶  $Z_1 = \{0, 1\}$
- ▶  $Z_2 = \{0, 1, 2, \dots, 9, A, B, C, D, E, F\}$
- ▶  $Z_3 = \{\alpha, \beta, \gamma, \dots, \omega\}$
- ▶  $Z_4 = \{\text{CR}, \text{LF}\}$



# Binärzeichen

**Binärzeichen** (engl: *binary element*, *binary digit*, *bit*): Jedes der Zeichen aus einem Vorrat / aus einer Menge von zwei Symbolen.

## Beispiel:

- ▶  $\mathcal{Z}_1 = \{0, 1\}$
- ▶  $\mathcal{Z}_2 = \{\text{high}, \text{low}\}$
- ▶  $\mathcal{Z}_3 = \{\text{rot}, \text{grün}\}$
- ▶  $\mathcal{Z}_4 = \{+, -\}$





# Alphabet

**Alphabet** (engl. *alphabet*): Ein in vereinbarter Reihenfolge geordneter Zeichenvorrat  $\mathcal{A} = \mathcal{Z}$

**Beispiel:**

- ▶  $\mathcal{A}_1 = \{0, 1, 2, \dots, 9\}$
- ▶  $\mathcal{A}_2 = \{ \text{So, Mo, Di, Mi, Do, Fr, Sa} \}$
- ▶  $\mathcal{A}_3 = \{ 'A', 'B', \dots, 'Z' \}$

**Numerischer Zeichensatz:** Zeichenvorrat aus Ziffern und/oder Sonderzeichen zur Darstellung von Zahlen

**Alphanumerischer Zeichensatz:** Zeichensatz aus (mindestens) den Dezimalziffern und den Buchstaben des gewöhnlichen Alphabets, meistens auch mit Sonderzeichen (Leerzeichen, Punkt, Komma usw.)



# Zeichenkette

**Zeichenkette** (engl. *string*): Eine Folge von Zeichen

**Wort** (engl. *word*): Eine Folge von Zeichen, die in einem gegebenen Zusammenhang als Einheit bezeichnet wird.

Worte mit 8 Bit werden als **Byte** bezeichnet.

**Stelle** (engl. *position*): Die Lage/Position eines Zeichens innerhalb einer Zeichenkette.

## Beispiel

▶ `s = H e l l o , w o r l d !`

# Darstellung von Zahlen und Zeichen: Übersicht

- ▶ Natürliche Zahlen (engl. *integer numbers*)
- ▶ Festkommazahlen (engl. *fixed point numbers*)
- ▶ Gleitkommazahlen (engl. *floating point numbers*)
  
- ▶ Aspekte der Textkodierung
- ▶ Adhoc-Kodierungen
- ▶ ASCII und ISO-8859-1
- ▶ Unicode
  
- ▶ Pointer (Referenzen, Maschinenadressen)

# Konzept der Zahl

- ▶ das Messen ist der Ursprung der Zahl
- ▶ als Abstraktion der Anzahl von Objekten
- ▶ die man abzählen kann
- ▶ Anwendung des Distributivgesetzes:

$$2 \text{ Äpfel} + 5 \text{ Äpfel} = 7 \text{ Äpfel}$$

$$2 \text{ Birnen} + 5 \text{ Birnen} = 7 \text{ Birnen}$$

...

- ▶  $2 + 5 = 7$

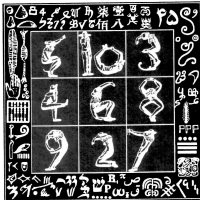


## Eigenschaften eines Zahlensystems

- ▶ Zahlenbereich: kleinste und größte darstellbare Zahl?
- ▶ Darstellung negativer Werte?
- ▶ Darstellung gebrochener Werte?
- ▶ Darstellung sehr großer Werte?
  
- ▶ Unterstützung von Rechenoperationen?
- ▶ Addition, Subtraktion, Multiplikation, Division, etc.
- ▶ Abgeschlossenheit unter diesen Operationen?
  
- ▶ Methode zur dauerhaften Speicherung/Archivierung?
- ▶ Sicherheit gegen Manipulation gespeicherter Werte?

## Literaturtipp

Georges Ifrah  
**Universal-  
 geschichte der  
 Zahlen**



(Georges Ifrah: Universalgeschichte der Zahlen 1998)



# Direkte Wahrnehmung vs. Zählen

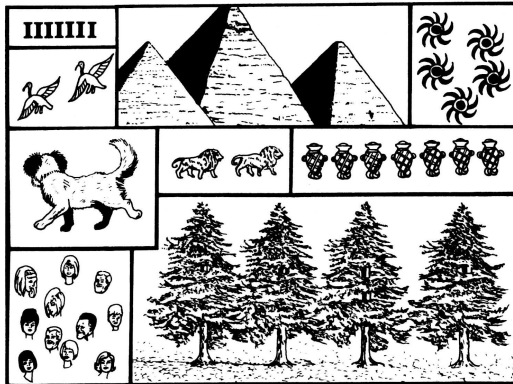


Abb. 1: Auf einen Blick können wir mit unserer direkten Zahlenwahrnehmung feststellen, ob eine Gesamtheit ein, zwei, drei oder vier Elemente umfaßt; Mengen, die größer sind, müssen wir meistens »zählen« – oder mit Hilfe des Vergleichs oder der gedanklichen Aufteilung in Teilmengen erfassen –, da unsere direkte Wahrnehmung nicht mehr ausreicht, exakte Angaben zu machen.



# Abstraktion: Verschiedene Symbole für eine Zahl

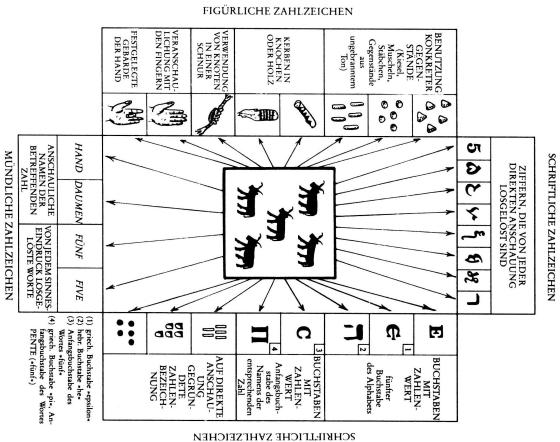
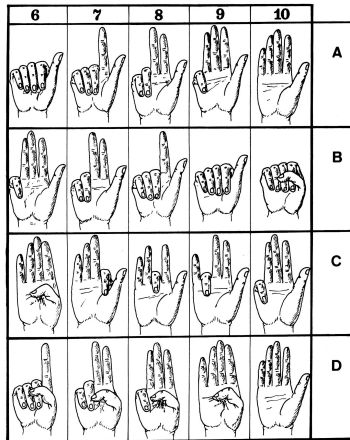
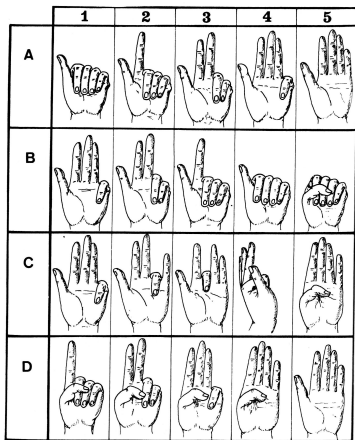


Abb. 1: Verbindungen einer ganzen Zahl (hier der Zahl 5) zugeordnete Symbole.

# Zählen mit den Fingern („digits“)



# Speicherung: Tonbörse: 15. Jh. v. Chr.

## Gegenstände, Hammel und Ziegen betreffend

- 21 Mutterschafe
- 6 weibliche Lämmer
- 8 erwachsene Hammel
- 4 männliche Lämmer
- 6 Mutterziegen
- 1 Bock
- (2) Jungziegen

Abb. 3: Eiförmige Tonbörse (46 mm × 62 mm × 50 mm), entdeckt in den Ruinen des Palastes von Nuzi (mesopotamische Stadt; ca. 15. Jh. v. Chr.).  
 (Harvard Semitic Museum, Cambridge.  
 Katalognummer SMN 1854)

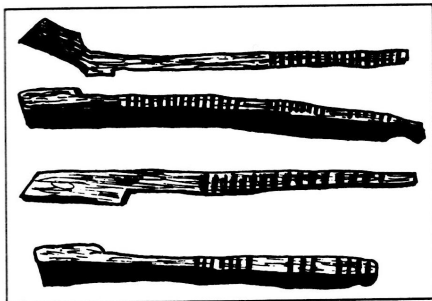


48 Tonkügelchen im Inneren: tamper-proof

# Speicherung: Kerbhölzer



*Abb. 58: Kerbhölzer aus Bäckereien in Frankreich, wie sie in kleinen Ortschaften auf dem Lande üblich waren.*



*Abb. 59: Englische Kerbhölzer aus dem 13. Jahrhundert.  
 (Sammlung Society of Antiquaries, London; Zeichnung nach Menninger 1957/58, II, 42)*

# Speicherung: Knotenschnüre

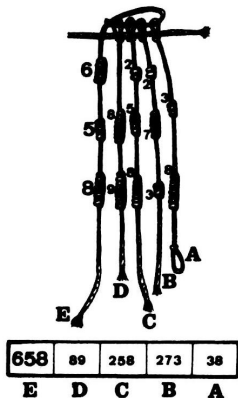


Abb. 66: Interpretation eines quipu: Die Zahl 658 auf der Schnur E ist gleich der Summe der Zahlen auf den Schnüren A, B, C und D. Dieses Bündel ist das erste an einem peruanischen quipu. (American Museum of Natural History, New York, B 8713; vgl. Leland Locke 1923)

## Rechnen: Römische Ziffern

- ▶ Ziffern: I=1, V=5, X=10, L=50, C=100, D=500, M=1000
- ▶ Werte eins bis zehn: I, II, III, IV, V, VI, VII, VIII, IX, X
- ▶ Position der Ziffern ist signifikant:
  - ▶ nach Größe der Ziffernsymbole sortiert, größere stehen links
  - ▶ andernfalls Abziehen der kleineren von der größeren Ziffer
  - ▶ IV=4, VI=6, XL=40, LXX=70, CM=900
- ▶ heute noch in Gebrauch: Jahreszahlen, Seitennummern, usw.
- ▶ Beispiele: MDCCCXIII=1813, MMIX=2009
- ▶ keine Symbole zur Darstellung großer Zahlen
- ▶ Rechenoperationen so gut wie unmöglich

# Römischer Abakus

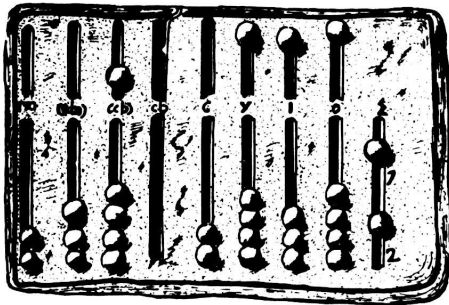


Abb. 87: Römischer Handabakus.  
 (Cabinet des Médailles, Bibliothèque Nationale Paris, br. 1925)


 $10^6$ 

 $10^5$ 

 $10^4$ 

 $10^3$ 

 $10^2$ 


10



1



# Römischer Abakus

Dagegen können im Rahmen einer entwickelten Stellenwertschrift nicht nur alle beliebigen Zahlen jeder Größenordnung mit einer beschränkten Anzahl von Ziffern dargestellt werden, sondern mit ihr kann auch sehr einfach gerechnet werden. Und eben deshalb ist unser Ziffernsystem eine der Grundlagen der geistigen Fähigkeiten der modernen Menschen.

Als Beweis dafür führen wir mit römischen Ziffern eine einfache Addition durch:

CCLXVI	266
MDCCCVII	1 807
DCL	650
MLXXX	1 080
MMMDCCLXXXIII	3 803

Ohne Übertragung auf unsere Zahlschrift wäre das sehr schwierig, wenn nicht unmöglich – und dabei handelt es sich doch bloß um eine Addition! Wie verhielte sich das erst bei einer Multiplikation oder gar bei einer Division? Mit diesen Ziffernsystemen kann nicht gerechnet werden, da ihre Grundziffern einen festgelegten Zahlenwert haben. Diese Ziffern sind keine Recheneinheiten, sondern Abkürzungen, mit denen Ergebnisse von Rechnungen festgehalten werden können, die mit Gegenständen auf der Rechentafel, dem Abakus oder dem Kugelbrett bereits gelöst worden waren.



# Das Stellenwertsystem („Radixdarstellung“)

- ▶ Wahl einer geeigneten Zahlenbasis  $b$  („Radix“)
  - ▶ 10: Dezimalsystem
  - ▶ 60: 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

# Einführung der Null: Babylon, 3 Jh. v. Chr.

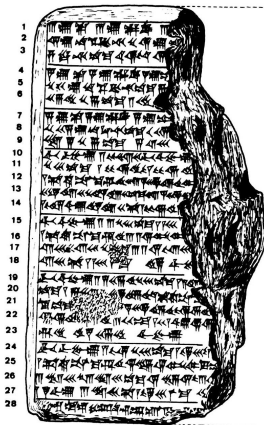
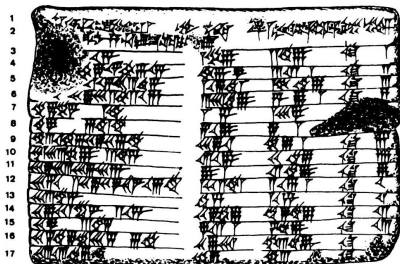


Abb. 289: Mathematische Tafel aus Uruk;  
 sie wurde bei Schwarzgrabungen gefunden  
 und stammt aus dem 2. oder 3. Jh. v. Chr.  
 Es handelt sich um eines der ältesten be-  
 kannten Zeugnisse für die Verwendung  
 der babylonischen Null.  
 (Musée du Louvre, Taf. AO 6484, Rück-  
 seite; Thureau-Dangin 1922, Nr. 33, Taf.  
 62; 1938, 76-81. Unveröffentl. Kopie d.  
 Verf.)

# Babylon: Beispiel mit Satz des Pythagoras



Transkription

Zeile 1	IB-SI-IL-TI SI-LI-IP-TIM	IB-SÁ SAG	IB-SÁ SI-LI-IP-TIM MU-BI-IM
2	NA-AS-SÁ-HU-Ú(SA) SAG-I ...-Ú		
3	15	1; 59	2; 49 KI 1
4	56; 7 58; 14 50; 6, 15	56; 7	3; 12, 1 KI 2
5	33; 41 15; 33, 45	1; 16, 41	1; 50, 49 KI 3
6	59; 31 32; 52, 16	3; 31, 49	5; 9, 1 KI 4
7	48; 54 1, 40	1; 5	1; 37 KI 5
8	47 6; 41, 40	5; 19	8; 1 KI 6
9	43; 11 56; 28, 20, 40	38; 11	59; 1 KI 7
10	41; 33, 59 3, 45	13; 19	20; 49 KI 8
11	38; 33, 36, 36	9; 1	12; 49 KI 9
12	35; 10, 2, 28, 27, 24, 16, 40	7; 22, 41	2; 16, 1 KI 10
13	33, 45	45	1; 15 KI 11
14	29; 21, 54, 2, 15	27; 59	48; 49 KI 12
15	27; 8, 3, 45	7; 12; 1	4; 49 KI 13
16	25; 48, 51, 35, 6, 40	26; 31	53; 49 KI 14
17	23; 13, 46, 50	50	KI 15

\*Leerstelle, die das Fehlen von Einheiten einer bestimmten Größenordnung bezeichnet.

Abb. 288: Rechentafel aus der Zeit um 1800–1700 v. Chr.; ihr Inhalt belegt, daß die babylonischen Mathematiker zur Zeit der 1. Dynastie bereits den «Satz des Pythagoras» kannten.

(Columbia University of New York, Tafel Plimpton 322; unveröffentl. Kopie d. Verf.; vgl. Neugebauer/Sachs 1945, 38–41, Taf. 25)

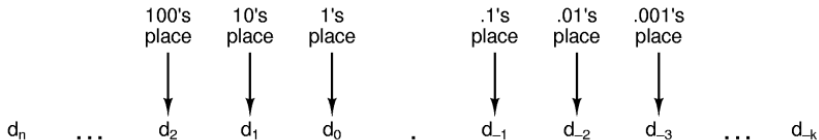


# Babylon: Beispiel Potenztabelle $100^i$



(Klaus von der Heide, Interaktives Skript T1, powersbabylon)

# Dezimalsystem



$$\text{Number} = \sum_{i=-k}^n d_i \times 10^i$$

- ▶ das im Alltag gebräuchliche Zahlensystem
- ▶ Einer, Zehner, Hunderter, Tausender, usw.
- ▶ Zehntel, Hundertstel, Tausendstel, usw.

# Dualsystem

- ▶ Stellenwertsystem zur Basis 2
- ▶ für gegebene Zahl ca. dreimal mehr Stellen als bei Basis 10
- ▶ für Menschen daher unbequem (besser Oktal- oder Hexadezimalschreibweise, s.u.)
  
- ▶ technisch besonders leicht zu implementieren
- ▶ weil nur zwei Zustände unterschieden werden müssen
  - ▶ z.B. zwei Spannungen, Ströme, Beleuchtungsstärken
  - ▶ s.o.: CMOS-Repräsentation
  - ▶ robust gegen Rauschen und Störungen
- ▶ einfache und effiziente Realisierung von Arithmetik

## Dualsystem: Potenztabelle

Stelle	Wert im Dualsystem	Wert im Dezimalsystem
$2^0$	1	1
$2^1$	10	2
$2^2$	100	4
$2^3$	1000	8
$2^4$	1 0000	16
$2^5$	10 0000	32
$2^6$	100 0000	64
$2^7$	1000 0000	128
$2^8$	1 0000 0000	256
$2^9$	10 0000 0000	512
$2^{10}$	100 0000 0000	1024
...	...	...



# Addition im Dualsystem

- ▶ funktioniert genau wie im Dezimalsystem
- ▶ Additionsmatrix:

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

- ▶ Beispiel:

$$\begin{array}{r}
 1011 \ 0011 \\
 + 0011 \ 1001 \\
 \hline
 \ddot{U} \ 11 \ 11 \\
 \hline
 1110 \ 1100
 \end{array}$$

## Multiplikation: 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 ist sehr einfach:

$$\begin{array}{r|ll}
 * & 0 & 1 \\
 \hline
 0 & 0 & 0 \\
 1 & 0 & 1
 \end{array}$$

## Multiplikation: Beispiel

$$\begin{array}{r}
 10110011 \quad * \quad 1101 \quad = \quad 179 \quad * \quad 13 \quad = \quad 2327 \quad = \quad 1001 \quad 0001 \quad 0111 \\
 \hline
 10110011 \quad 1 \\
 10110011 \quad 1 \\
 00000000 \quad 0 \\
 10110011 \quad 1 \\
 \hline
 \ddot{U} \quad 11 \quad 1111 \\
 \hline
 100100010111
 \end{array}$$



# Oktalsystem

- ▶ Basis 8
- ▶ Zeichensatz ist  $\{0, 1, 2, 3, 4, 5, 6, 7\}$
- ▶ C-Schreibweise mit führender Null als Präfix:

$$0001 = 1_{10}$$

$$0013 = 11_{10} = 1 \cdot 8 + 3$$

$$0375 = 253_{10} = 3 \cdot 64 + 7 \cdot 8 + 5$$

usw.

Hinweis: also führende Null in C für Dezimalzahlen unmöglich

- ▶ für Menschen leichter lesbar als Dualzahlen
- ▶ Umwandlung aus/vom Dualsystem durch Zusammenfassen bzw. Ausschreiben von je drei Bits

# Hexadezimalsystem

- ▶ Basis 16
- ▶ Zeichensatz ist  $\{0, 1, 2, \dots, 9, A, B, C, D, E, F\}$
- ▶ C-Schreibweise mit Präfix 0x, Klein- oder Grossbuchstaben

0x00000001 (dezimal 1)

0x000000fe (dezimal 254:  $15 \cdot 16 + 14$ )

0x0000ffff (dezimal 65535)

0xcafebabe (erstes Wort in Java Class-Dateien)

usw.

- ▶ viel leichter lesbar als entsprechende Dualzahl
- ▶ Umwandlung aus/vom Dualsystem: Zusammenfassen bzw. Ausschreiben von je vier Bits:

$0x0=0000$ ,  $0x1=0001$ ,  $0x2=0010$ ,  $\dots$ ,  $0x9=1001$ ,

$0xA=1010$ ,  $0xB=1011$ ,  $0xC=1100$ ,  $0xD=1101$ ,  $0xE=1110$ ,  $0xF=1111$

## Beispiel: Darstellungen der Zahl 2001

Binary      1      1      1      1      1      0      1      0      0      0      1

$$1 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

1024 + 512 + 256 + 128 + 64 + 0 + 16 + 0 + 0 + 0 + 1

Octal        3      7      2      1

$$3 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$$

1536 + 448 + 16 + 1

Decimal     2      0      0      1

$$2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$$

2000 + 0 + 0 + 1

Hexadecimal 7      D      1                      .

$$7 \times 16^2 + 13 \times 16^1 + 1 \times 16^0$$

1792 + 208 + 1

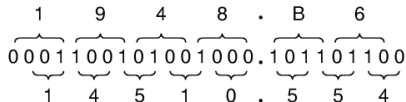
# Umrechnung Dual-/Oktal-/Hexadezimalsystem

## Example 1

Hexadecimal

Binary

Octal

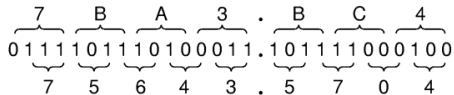


## Example 2

Hexadecimal

Binary

Octal



- ▶ Gruppieren von jeweils 3 bzw. 4 Bits
- ▶ bei Festkomma vom Dezimalpunkt aus nach außen

# Umrechnung zwischen verschiedenen Basen

- ▶ Menschen rechnen im Dezimalsystem
- ▶ Winkel- und Zeitangaben auch im Sedezimalsystem
- ▶ Digitalrechner nutzen (meistens) Dualsystem
  
- ▶ Algorithmen zur Umrechnung notwendig
- ▶ Exemplarisch Vorstellung von drei Varianten:
  - ▶ vorberechnete Potenztabellen
  - ▶ Divisionsrestverfahren
  - ▶ Horner-Schema



# Umwandlung über Potenztabellen

Vorgehensweise für Integerzahlen:

- ▶ Subtraktion des größten Vielfachen einer Potenz des Zielsystems (gemäß der vorberechneten Potenztabelle) von der umzuwandelnden Zahl
- ▶ Notation dieses größten Vielfachen (im Zielsystem)
- ▶ Subtraktion wiederum des größten Vielfachen vom verbliebenen Rest
- ▶ Addition des zweiten Vielfachen zum ersten
- ▶ Wiederholen, bis Rest = 0

# Potenztabellen Dual/Dezimal

Stelle	Wert	Stelle	Wert im Dualsystem
$2^0$	1	$10^0$	1
$2^1$	2	$10^1$	1010
$2^2$	4	$10^2$	110 0100
$2^3$	8	$10^3$	111 1101 0000
$2^4$	16	$10^4$	10 0111 0001 0000
$2^5$	32	$10^5$	0x186A0
$2^6$	64	$10^6$	0xF4240
$2^7$	128	$10^7$	0x989680
$2^8$	256	$10^8$	0x5F5E100
$2^9$	512	$10^9$	0x369ACA00
$2^{10}$	1024	$10^{10}$	
...			

# Beispiel

## Beispiel: Umwandlung Dezimal- in Dualzahl:

(Verwendung von Potenztabelle 1.3.1.1/1)

Annahme:  $z = (163)_{10}$

$$\begin{array}{r}
 163 \\
 \underline{-128} \quad (2^7) \quad 1000.0000 \\
 35 \\
 \underline{-32} \quad (2^5) \quad + \quad 10.0000 \\
 3 \\
 \underline{-2} \quad (2^1) \quad + \quad 10 \\
 1 \\
 \underline{-1} \quad (2^0) \quad + \quad 1 \\
 \hline
 0 \qquad \qquad \qquad 1010.0011
 \end{array}$$

$$(163)_{10} \longleftrightarrow (1010.0011)_2$$

## Beispiel

### Beispiel: Umformung Dual- in Dezimalzahl:

gegeben:  $z = (1010.0011)_2$

$$\begin{array}{r}
 1010.0011 \\
 - \quad \underline{110.0100} \quad (100)_{10} = 1 \cdot 100 = 1 \cdot 10^2 \\
 0011.1111 \\
 - \quad \underline{11.1100} \quad (60)_{10} = 6 \cdot 10 = 6 \cdot 10^1 \\
 \quad \quad 11 \\
 - \quad \quad \underline{11} \quad (3)_{10} = 3 \cdot 1 = 3 \cdot 10^0 \\
 \quad \quad \quad 0 \quad (163)_{10}
 \end{array}$$



## Divisionsrestverfahren

- ▶ Division der umzuwandelnden Zahl im Ausgangssystem durch die Basis des Zielsystems
- ▶ Erneute Division des ganzzahligen Ergebnisses (ohne Rest) durch die Basis des Zielsystems, bis kein ganzzahliger Divisionsrest mehr bleibt
- ▶ Beispiel:

$$\begin{array}{rcl}
 163 : 2 = 81 & \text{Rest} & 1 \\
 81 : 2 = 40 & \text{Rest} & 1 \\
 40 : 2 = 20 & \text{Rest} & 0 \\
 20 : 2 = 10 & \text{Rest} & 0 \\
 10 : 2 = 5 & \text{Rest} & 0 \\
 5 : 2 = 2 & \text{Rest} & 1 \\
 2 : 2 = 1 & \text{Rest} & 0 \\
 1 : 2 = 0 & \text{Rest} & 1
 \end{array}
 \left. \vphantom{\begin{array}{rcl} 163 : 2 = 81 \\ 81 : 2 = 40 \\ 40 : 2 = 20 \\ 20 : 2 = 10 \\ 10 : 2 = 5 \\ 5 : 2 = 2 \\ 2 : 2 = 1 \\ 1 : 2 = 0 \end{array}} \right\} \uparrow$$

$$(163)_{10} \longleftrightarrow (1010.0011)_2$$

## Beispiel

### Beispiel: Umwandlung Dual- in Dezimalzahl:

(wie oben)

$$\begin{array}{rcl}
 (1010.0011)_2 : (1010)_2 = 1.0000 & \text{Rest} & (11)_2 \hat{=} (3)_{10} \\
 (1.0000)_2 : (1010)_2 = & 1 & \text{Rest} & (110)_2 \hat{=} (6)_{10} \\
 (1)_2 : (1010)_2 = & 0 & \text{Rest} & (1)_2 \hat{=} (1)_{10}
 \end{array} \left. \vphantom{\begin{array}{rcl} (1010.0011)_2 : (1010)_2 = 1.0000 \\ (1.0000)_2 : (1010)_2 = 1 \\ (1)_2 : (1010)_2 = 0 \end{array}} \right\} \uparrow$$

$$(1010.0011)_2 \longleftrightarrow (163)_{10}$$

(Hinweis: Division in Basis  $b$  nächste Woche)



# Horner-Schema

- ▶ Darstellung einer Potenzsumme durch ineinander verschachtelte Faktoren

$$|z| = \sum_{i=0}^{n-1} a_i \cdot b^i = (\dots (((a_{n-1} \cdot b + a_{n-2})) \cdot b + a_{n-3}) \cdot b + \dots + a_1) \cdot b + a_0$$

Vorgehensweise:

- ▶ Darstellung der umzuwandelnden Zahl im Horner-Schema
- ▶ Durchführung der auftretenden Multiplikationen und Additionen im Zielsystem



## Beispiel

gegeben:  $(163)_{10}$

$$163 = (1 \cdot 10 + 6) \cdot 10 + 3$$

Umsetzung von Faktoren und Summanden ins Zielzahlensystem:

$$(10)_{10} \longleftrightarrow (1010)_2$$

$$(1)_{10} \longleftrightarrow (0001)_2$$

$$(6)_{10} \longleftrightarrow (0110)_2$$

$$(3)_{10} \longleftrightarrow (0011)_2$$

Durchführung der arithmetischen Operation

$$\begin{array}{r}
 0001 \cdot 1010 = 1010 \\
 + \quad 0110 \\
 \hline
 1.0000 \cdot 1010 = 1010.0000 \\
 + \quad \quad 0011 \\
 \hline
 1010.0011
 \end{array}$$

# Beispiel

Rückumwandlung von Dual- in Dezimalzahl

$$(1010.0011)_2 = ((((((1 \cdot 10_2) + 0) \cdot 10_2 + 1) \cdot 10_2 + 0) \cdot 10_2 + 0) \cdot 10_2 + 0) \cdot 10_2 + 1) \cdot 10_2 + 1$$

Umsetzung von Faktoren und Summanden

$$\begin{aligned} (0)_2 &\longleftrightarrow (0)_{10} \\ (1)_2 &\longleftrightarrow (1)_{10} \\ (10)_2 &\longleftrightarrow (2)_{10} \end{aligned}$$

# Beispiel

Berechnung:

$$\begin{array}{r}
 1 \cdot 2 = 2 \\
 + \frac{0}{2} \cdot 2 = 4 \\
 + \frac{1}{5} \cdot 2 = 10 \\
 + \frac{0}{10} \cdot 2 = 20 \\
 + \frac{0}{20} \cdot 2 = 40 \\
 + \frac{0}{40} \cdot 2 = 80 \\
 + \frac{1}{81} \cdot 2 = 162 \\
 \hline
 \hline
 163
 \end{array}$$

## Zahlenbereich bei fester Wortlänge

Anzahl der Bits		Zahlenbereich jeweils von 0 bis $(2^n - 1)$
4-bit	$2^4 =$	16
8-bit	$2^8 =$	256
10-bit	$2^{10} =$	1 024
12-bit	$2^{12} =$	4 096
16-bit	$2^{16} =$	65 536
20-bit	$2^{20} =$	1 048 576
24-bit	$2^{24} =$	16 777 216
32-bit	$2^{32} =$	4 294 967 296
48-bit	$2^{48} =$	281 474 976 710 656
64-bit	$2^{64} =$	18 446 744 073 709 551 616

# Präfixe

Für die vereinfachte Schreibweise von großen bzw. sehr kleinen Werten ist die Präfixangabe als Abkürzung von Zehnerpotenzen üblich. Beispiele:

- ▶ Lichtgeschwindigkeit:  $300\,000\text{ km/s} = 30\text{ cm/ns}$
- ▶ Ruheenergie des Elektrons:  $0.51\text{ MeV}$
- ▶ Strukturbreite heutiger Mikrochips:  $45\text{ nm}$
- ▶ usw.

Es gibt entsprechende Präfixe auch für das Dualsystem. Dazu werden Vielfache von  $2^{10} = 1024 \approx 1000$  verwendet.

## Präfixe für Einheiten im Dezimalsystem

Faktor	Name	Symbol	Faktor	Name	Symbol
$10^{24}$	yotta	Y	$10^{-24}$	yocto	y
$10^{21}$	zetta	Z	$10^{-21}$	zepto	z
$10^{18}$	exa	E	$10^{-18}$	atto	a
$10^{15}$	peta	P	$10^{-15}$	femto	f
$10^{12}$	tera	T	$10^{-12}$	pico	p
$10^9$	giga	G	$10^{-9}$	nano	n
$10^6$	mega	M	$10^{-6}$	micro	$\mu$
$10^3$	kilo	K	$10^{-3}$	milli	m
$10^2$	hecto	h	$10^{-2}$	centi	c
$10^1$	deka	da	$10^{-1}$	dezi	d



## Präfixe für Einheiten im Dualsystem

Faktor	Name	Symbol	Langname
$2^{60}$	exbi	Ei	exabinary
$2^{50}$	pebi	Pi	petabinary
$2^{40}$	tebi	Ti	terabinary
$2^{30}$	gibi	Gi	gigabinary
$2^{20}$	mebi	Mi	megabinary
$2^{10}$	kibi	Ki	kilobinary

Beispiele:

1 kibibit = 1 024 bit

1 kilobit = 1 000 bit

1 megibit = 1 048 576 bit

1 gibibit = 1 073 741 824 bit

(IEC-60027-2, Letter symbols to be used in electrical technology)

## Präfixe für Einheiten im Dualsystem

In der Praxis werden die offiziellen Präfixe nicht immer sauber verwendet. Meistens ergibt sich die Bedeutung aber aus dem Kontext. Bei Speicherbausteinen sind Zweierpotenzen üblich, bei Festplatten dagegen die dezimale Angabe.

- ▶ DRAM-Modul mit 1 GB Kapazität: gemeint sind  $2^{30}$  Bytes
- ▶ Flash-Speicherkarte 4 GB Kapazität: gemeint sind  $2^{32}$  Bytes
- ▶ Festplatte mit Angabe 1 TB Kapazität: typisch  $10^{12}$  Bytes
- ▶ die tatsächliche angezeigte verfügbare Kapazität ist oft geringer, weil das jeweilige Dateisystem Platz für seine eigenen Verwaltungsinformationen belegt.



# Festkommadarstellung

Darstellung von **gebrochenen Zahlen** als Erweiterung des Stellenwertsystems durch Erweiterung des Laufindex zu negativen Werten:

$$\begin{aligned}
 |z| &= \sum_{i=0}^{n-1} a_i \cdot b^i + \sum_{i=-\infty}^{i=-1} a_i \cdot b^i \\
 &= \sum_{i=-\infty}^{n-1} a_i \cdot b^i
 \end{aligned}$$

mit  $a_i \in N$  und  $0 \leq a_i < b$ .

- ▶ Der erste Summand bezeichnet den ganzzahligen Anteil, während der zweite Summand für den gebrochenen Anteil steht.



## Nachkommastellen im Dualsystem

$$2^{-1} = 0.5$$

$$2^{-2} = 0.25$$

$$2^{-3} = 0.125$$

$$2^{-4} = 0.0625$$

$$2^{-5} = 0.03125$$

$$2^{-6} = 0.015625$$

$$2^{-7} = 0.0078125$$

...

- ▶ alle Dualbrüche sind im Dezimalsystem exakt darstellbar (d.h. mit endlicher Wortlänge)

## Beispiel: Umrechnung dezimal 0.3 nach dual

Betrachtung von gebrochenen Zahlen im Dualsystem:

$2^{-1} = 0,5$	$2^{-7} = 0,0078125$
$2^{-2} = 0,25$	$2^{-8} = 0,00390625$
$2^{-3} = 0,125$	$2^{-9} = 0,001953125$
$2^{-4} = 0,0625$	$2^{-10} = 0,0009765625$
$2^{-5} = 0,03125$	$2^{-11} = 0,00048828125$
$2^{-6} = 0,015625$	$2^{-12} = 0,000244140625$

Tab. 1.3.1.2/1

**Beispiel:**

$$\begin{aligned}
 (0,3)_{10} &= 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} \\
 &\quad + 1 \cdot 2^{-6} + \dots \\
 &= 2^{-2} + 2^{-5} + 2^{-6} + 2^{-9} + \dots \\
 &= (0,01001)_{2}
 \end{aligned}$$



## Beispiel: Dezimalbrüche, eine Nachkommastelle

- ▶ gebrochene Zahlen können je nach Wahl der Basis evtl. nur als unendliche periodische Brüche dargestellt werden.
- ▶ insbesondere: viele endliche Dezimalbrüche erfordern im Dualsystem unendliche periodische Brüche.

B=10	B=2	B=2	B=10
0,1	$0,00011$	0,001	0,125
0,2	$0,0011$	0,010	0,250
0,3	$0,01001$	0,011	0,375
0,4	$0,0110$	0,100	0,5
0,5	$0,1$	0,101	0,625
0,6	$0,1001$	0,110	0,750
0,7	$0,10110$	0,111	0,875
0,8	$0,1100$		
0,9	$0,11100$		



# Darstellung negativer Zahlen

Drei gängige Varianten zur Darstellung negativer Zahlen:

- ▶ Betrag und Vorzeichen
- ▶ Exzess-Kodierung (Offset-basiert)
- ▶ **Komplementdarstellung**
  
- ▶ **Integerrechnung häufig im Zweierkomplement**
- ▶ **Gleitkommadarstellung mit Betrag und Vorzeichen**

# Darstellung negativer Zahlen: Beispiele

<b>N</b> decimal	<b>N</b> binary	<b>-N</b> signed mag.	<b>-N</b> 1's compl.	<b>-N</b> 2's compl.	<b>-N</b> excess 128
1	00000001	10000001	11111110	11111111	01111111
2	00000010	10000010	11111101	11111110	01111110
3	00000011	10000011	11111100	11111101	01111101
4	00000100	10000100	11111011	11111100	01111100
5	00000101	10000101	11111010	11111011	01111011
6	00000110	10000110	11111001	11111010	01111010
7	00000111	10000111	11111000	11111001	01111001
8	00001000	10001000	11110111	11111000	01111000
9	00001001	10001001	11110110	11110111	01110111
10	00001010	10001010	11110101	11110110	01110110
20	00010100	10010100	11101011	11101100	01101100
30	00011110	10011110	11100001	11100010	01100010
40	00101000	10101000	11010111	11011000	01011000
50	00110010	10110010	11001101	11001110	01001110
60	00111100	10111100	11000011	11000100	01000100
70	01000110	11000110	10111001	10111010	00111010
80	01010000	11010000	10101111	10110000	00110000
90	01011010	11011010	10100101	10100110	00100110
100	01100100	11011010	10011011	10011100	00011100
127	01111111	11111111	10000000	10000001	00000001
128	Nonexistent	Nonexistent	Nonexistent	10000000	00000000

## Betrag und Vorzeichen

- ▶ Auswahl eines Bits als Vorzeichenbit
- ▶ meistens das MSB (engl. *most significant bit*)
- ▶ restliche Bits als Dualzahl interpretiert
- ▶ Beispiel für 4-bit Wortbreite:

0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4

...

0111	+7	1111	-7
------	----	------	----

- ▶ doppelte Kodierung der Null: +0, -0
- ▶ Nachteil: Rechenwerke für Addition/Subtraktion aufwendig

## Exzess-Kodierung (Offset-basiert)

- ▶ einfache Um-Interpretation der Binärkodierung

$$z = Z - \text{offset}$$

- ▶ mit  $z$  vorzeichenbehafteter Wert,  $Z$  binäre Ganzzahl,
- ▶ beliebig gewählter Offset
- ▶ Null wird also nicht mehr durch  $000\dots 0$  dargestellt
- ▶ Größenvergleich zweier Zahlen bleibt einfach
- ▶ Anwendung: Exponenten im IEEE-754 Gleitkommaformat
- ▶ und für einige Audioformate





## Exzess-Kodierung: Beispiel

Bitmuster	Binärkode	Exzess-8	Exzess-6	( $z = Z - \text{offset}$ )
0000	0	-8	-6	
0001	1	-7	-5	
0010	2	-6	-4	
0011	3	-5	-3	
0100	4	-4	-2	
0101	5	-3	-1	
0110	6	-2	0	
0111	7	-1	1	
1000	8	0	2	
1001	9	1	3	
1010	10	2	4	
1011	11	3	5	
1100	12	4	6	
1101	13	5	7	
1110	14	6	8	
1111	15	7	9	



## *b*-Komplement: Definition

Definition: das *b*-Komplement einer Zahl *z* ist

$$K_b(z) = b^n - z, \quad \text{für } z \neq 0$$

$$K_b(z) = 0, \quad \text{für } z = 0$$

- ▶ *b*: die Basis (des Stellenwertsystems)
- ▶ *n*: Anzahl der zu berücksichtigenden Vorkommastellen
- ▶ mit anderen Worten:  $K_b(z) + z = b^n$
  
- ▶ Dualsystem: 2-Komplement
- ▶ Dezimalsystem: 10-Komplement



## *b*-Komplement: Beispiele

$$b = 10, n = 4 \quad K_{10}(3763)_{10} = 10^4 - 3763 = 6237_{10}$$

$$b = 10, n = 2 \quad K_{10}(0.3763)_{10} = 10^2 - 0.3763 = 99.6237_{10}$$

$$b = 10, n = 0 \quad K_{10}(0.3763)_{10} = 10^0 - 0.3763 = 0.6237_{10}$$

$$b = 2, n = 2 \quad K_2(10.01)_2 = 2^2 - 10.01_2 = 01.11_2$$

$$b = 2, n = 8 \quad K_2(10.01)_2 = 2^8 - 10.01_2 = 11111101.11_2$$





## $(b - 1)$ -Komplement: Trick

$$K_{b-1}(z) = b^n - b^{-m} - z, \quad \text{für } z \neq 0$$

- ▶ Im Fall  $m = 0$  gilt offenbar  $K_b(z) = K_{b-1}(z) + 1$
- ▶ Trick: das  $(b - 1)$ -Komplement kann sehr einfach berechnet werden. Es werden einfach die einzelnen Bits/Ziffern invertiert.

- |                  |                          |                    |
|------------------|--------------------------|--------------------|
| ▶ Dualsystem:    | 1-Komplement             | 1100 1001          |
|                  | alle Bits invertieren    | 0011 0110          |
| ▶ Dezimalsystem: | 9-Komplement             | 24453              |
|                  | alle Ziffern invertieren | 75546              |
|                  | 0-9 1-8 2-7 3-6 4-5      |                    |
|                  | Summe:                   | 99999 = 100000 - 1 |

## 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:

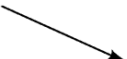

$$x - y = x + K_b(y)$$

- ▶ Subtraktion kann also durch Addition des  $b$ -Komplements ersetzt werden
- ▶ und für Integerzahlen gilt außerdem

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

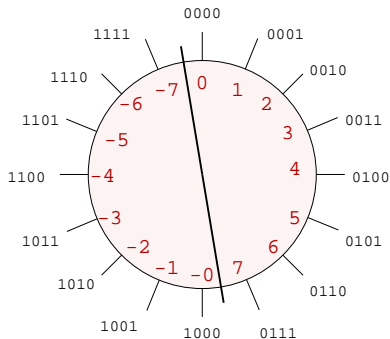
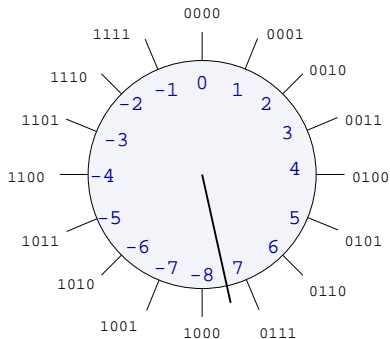
# Subtraktion mit Einer- und Zweierkomplement

- ▶ Subtraktion ersetzt durch Addition des Komplements

<u>Decimal</u>	<u>1's complement</u>	<u>2's complement</u>
$\begin{array}{r} 10 \\ + (-3) \\ \hline \end{array}$	$\begin{array}{r} 00001010 \\ 11111100 \\ \hline \end{array}$	$\begin{array}{r} 00001010 \\ 11111101 \\ \hline \end{array}$
$+7$	$1 \ 00000110$	$1 \ 00000111$
	 $\begin{array}{r} \text{carry } 1 \\ \hline \end{array}$	 discarded
	$00000111$	

# Veranschaulichung: Zahlenkreis

Beispiel für  $w$ -bit



► links: Zweierkomplement    rechts: Betrag und Vorzeichen



# Gleitkommaformat

Wie kann man „wissenschaftliche“ Zahlen darstellen?

- ▶ Masse der Sonne  $1.989 \cdot 10^{30}$  kg
- ▶ Masse eines Elektrons 0.00000 00000 00000 00016 g
- ▶ Anzahl der Atome pro Mol 6023 00000 00000 00000 00000
- ...

Darstellung im Stellenwertsystem?

- ▶ gleichzeitig sehr große und sehr kleine Zahlen notwendig
- ▶ entsprechend hohe Zahl der Vorkomma- und Nachkommastellen
- ▶ durchaus möglich (Java3D: 256-bit Koordinaten)
- ▶ aber normalerweise sehr unpraktisch
- ▶ typische Meßwerte haben nur ein paar Stellen Genauigkeit



# Gleitkomma: Motivation

Grundidee: **halblogarithmische Darstellung einer Zahl:**

- ▶ Vorzeichen (+1 oder -1)
- ▶ *Mantisse* als normale Zahl im Stellenwertsystem
- ▶ *Exponent* zur Angabe der Größenordnung

$$z = \text{sign} \cdot \text{mantisse} \cdot \text{basis}^{\text{exponent}}$$

- ▶ handliche Wertebereiche für Mantisse und Exponent
- ▶ arithmetische Operationen sind effizient umsetzbar
- ▶ Wertebereiche für ausreichende Genauigkeit wählen

(Hinweis: rein logarithmische Darstellung wäre auch möglich, aber Addition/Subtraktion sind dann sehr aufwendig.)

# Gleitkomma: Dezimalsystem

$$z = (-1)^s \cdot m \cdot 10^e$$

- ▶  $s$                       Vorzeichenbit
  - ▶  $m$                       Mantisse als Festkomma-Dezimalzahl
  - ▶  $e$                       Exponent als ganze Dezimalzahl
- 
- ▶ Schreibweise in C/Java: Vorzeichen Mantisse E Exponent
 

6.023E23	$6.023 \cdot 10^{23}$ (Avogadro-Zahl)
1.6E-19	$1.6 \cdot 10^{-19}$ (Elementarladung des Elektrons)

# Gleitkomma: Beispiel für Zahlenbereiche

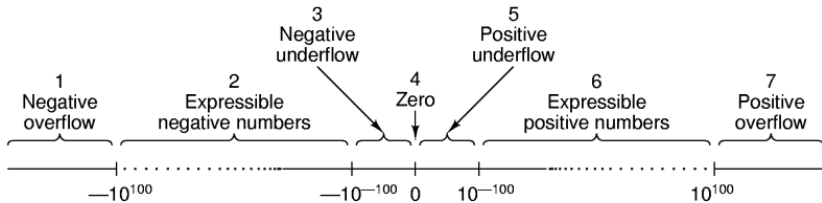
Digits in fraction	Digits in exponent	Lower bound	Upper bound
3	1	$10^{-12}$	$10^9$
3	2	$10^{-102}$	$10^{99}$
3	3	$10^{-1002}$	$10^{999}$
3	4	$10^{-10002}$	$10^{9999}$
4	1	$10^{-13}$	$10^9$
4	2	$10^{-103}$	$10^{99}$
4	3	$10^{-1003}$	$10^{999}$
4	4	$10^{-10003}$	$10^{9999}$
5	1	$10^{-14}$	$10^9$
5	2	$10^{-104}$	$10^{99}$
5	3	$10^{-1004}$	$10^{999}$
5	4	$10^{-10004}$	$10^{9999}$
10	3	$10^{-1009}$	$10^{999}$
20	3	$10^{-1019}$	$10^{999}$

## Gleitkomma: Historie

- ▶ 1937      Zuse: Z1 mit 22-bit Gleitkomma-Datenformat
- ▶ ...
- ▶ 195x      Verbreitung von Gleitkomma-Darstellung  
für numerische Berechnungen
- ▶ 1980      Intel 8087: erster Koprozessor-Chip,  
ca. 45.000 Transistoren, ca. 50k FLOP/s
- ▶ 1985      IEEE-754 Standard für Gleitkomma
- ▶ 1989      Intel 486 mit integriertem Koprozessor
- ▶ 1995      Java-Spezifikation fordert IEEE-754
- ▶ 1996      ASCI-RED: 1 TFLOPS    (9152x PentiumPro)
- ▶ 2008      Roadrunner: 1 PFLOPS    (12960x Cell)
- ▶ ...

(FLOPS := floating-point operations per second)

# Gleitkomma: Zahlenbereiche



- ▶ Darstellung üblicherweise als Betrag+Vorzeichen
- ▶ negative und positive Zahlen gleichberechtigt (symmetrisch)
- ▶ separate Darstellung für den Wert Null
- ▶ sieben Zahlenbereiche: siehe Bild
- ▶ relativer Abstand benachbarter Zahlen bleibt ähnlich (vgl. dagegen Integer:  $0/1, 1/2, 2/3, \dots, 65535/65536, \dots$ )

# Gleitkomma: Normalisierte Darstellung

$$z = (-1)^s \cdot m \cdot 10^e$$

- ▶ diese Darstellung ist bisher nicht eindeutig:

$$123 \cdot 10^0 = 12.3 \cdot 10^1 = 1.23 \cdot 10^2 = 0.123 \cdot 10^3 = \dots$$

## normalisierte Darstellung

- ▶ Exponent anpassen, bis Mantisse im Bereich  $1 \leq m < b$  liegt
- ▶ Darstellung ist dann eindeutig
- ▶ im Dualsystem: erstes Vorkommabit ist dann also 1, muss also nicht gespeichert werden
- ▶ evtl. zusätzlich sehr kleine Zahlen nicht-normalisiert

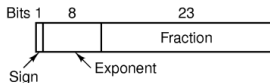
# IEEE-754 Standard

bis 1985 ein Wildwuchs von Gleitkomma-Formaten:

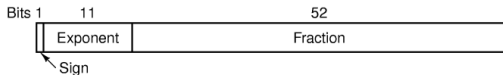
- ▶ unterschiedliche Anzahl Bits in Mantisse und Exponent
- ▶ Exponent mit Basis 2, 10, oder 16
- ▶ diverse Algorithmen zur Rundung
- ▶ jeder Hersteller mit eigener Variante
  
- ▶ Numerische Algorithmen nicht portabel
  
- ▶ 1985: Publikation des Standards IEEE-754 zur Vereinheitlichung
- ▶ klare Regeln, auch für Rundungsoperationen
- ▶ große Akzeptanz, mittlerweile der universale Standard



## IEEE-754: *float* und *double*



(a)



- ▶ 32-bit-Format: einfache Genauigkeit (*single precision*, *float*)
- ▶ 64-bit-Format: doppelte Genauigkeit (*double precision*, *double*)
- ▶ Mantisse als normalisierte Dualzahl,  $1 \leq m < 2$
- ▶ Exponent in Exzess-127 bzw. Exzess-1023 Kodierung
- ▶ einige Sonderwerte: Null (+0, -0), NaN, Infinity

# IEEE-754: Zahlenbereiche

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	-126 to +127	-1022 to +1023
Smallest normalized number	$2^{-126}$	$2^{-1022}$
Largest normalized number	approx. $2^{128}$	approx. $2^{1024}$
Decimal range	approx. $10^{-38}$ to $10^{38}$	approx. $10^{-308}$ to $10^{308}$
Smallest denormalized number	approx. $10^{-45}$	approx. $10^{-324}$

# Matlab-Demo: demoieee754

Zahl

178.125

IEEE-754

short real  
 long real

± Exponent implizit      Mantisse

0 10000110 1. 011001000100000000000000

- ▶ Zahlenformat wählen (float=short real, double=long real)
- ▶ Dezimalzahl in oberes Textfeld eingeben
- ▶ Mantisse/Exponent/Vorzeichen in unteres Textfeld eingeben
- ▶ andere Werte werden jeweils aktualisiert



## Beispiele: float

- ▶ 1-bit Vorzeichen 8-bit Exponent (Exzess-127), 23-bit Mantisse

$$z = (-1)^s \cdot 2^{(eeee\ eeee-127)} \cdot 1.mmmm\ mmmm\ mmmm \dots mmm$$

- ▶ 1 1000 0000 1110 0000 0000 0000 0000 000

$$\begin{aligned} z &= -1 \cdot 2^{(128-127)} \cdot (1 + 0.5 + 0.25 + 0.125 + 0) \\ &= -1 \cdot 2 \cdot 1.875 = -3.750 \end{aligned}$$

- ▶ 0 1111 1110 0001 0011 0000 0000 0000 000

$$\begin{aligned} z &= +1 \cdot 2^{(254-127)} \cdot (1 + 2^{-4} + 2^{-7} + 2^{-8}) \\ &= 2^{127} \cdot 1.07421875 = 1.953965 \cdot 10^{38} \end{aligned}$$



## Beispiele: float

$$z = (-1)^s \cdot 2^{(eeee\ eeee-127)} \cdot 1.mmmm\ mmmm\ mmmm \dots mmm$$

- ▶ 1 0000 0001 0000 0000 0000 0000 0000 0000 000

$$\begin{aligned} z &= -1 \cdot 2^{(1-127)} \cdot (1 + 0 + 0 + \dots + 0) \\ &= -1 \cdot 2^{-126} \cdot 1.0 = -1.1755 \cdot 10^{-38} \end{aligned}$$

- ▶ 0 0111 1111 0000 0000 0000 0000 0000 0000 001

$$\begin{aligned} z &= +1 \cdot 2^{(127-127)} \cdot (1 + 2^{-23}) \\ &= 1 \cdot (1 + 0.0000001) = 1.0000001 \end{aligned}$$



## BitsToFloat.java

```

public static void main( String[] args ) {
    p( "1", "01111111", "000000000000000000000000" );
}
public void p( String s, String e, String m ) {
    int    sign = (Integer.parseInt( s, 2 ) & 0x1) << 31;
    int exponent = (Integer.parseInt( e, 2 ) & 0xFF) << 23;
    int mantisse = (Integer.parseInt( m, 2 ) & 0x007FFFFFFF);
    int bits = sign | exponent | mantisse;
    float f = Float.intBitsToFloat( bits );
    System.out.println( dumpIntBits(bits) + " " + f );
}
public String dumpIntBits( int i ) {
    StringBuffer sb = new StringBuffer();
    for( int mask=0x80000000; mask != 0; mask = mask >>> 1 ) {
        sb.append( ((i & mask) != 0) ? "1" : "0" );
    }
    return sb.toString();
}
    
```





## Gleitkomma: Addition, Subtraktion

Addition von Gleitkommazahlen  $y = a_1 + a_2$

- ▶ Skalierung des betragsmäßig kleineren Summanden
- ▶ Erhöhen des Exponenten, bis  $e_1 = e_2$  gilt
- ▶ gleichzeitig entsprechendes Skalieren der Mantisse
- ▶ Achtung: dabei verringert sich die effektive Genauigkeit des kleineren Summanden
  
- ▶ anschließend Addition/Subtraktion der Mantissen
- ▶ ggf. Normalisierung des Resultats
  
- ▶ Beispiele in den Übungen

## Gleitkomma-Addition: Beispiel

$$a = 9.725 \cdot 10^7; b = 3.016 \cdot 10^6$$

$$y = (a + b)$$

$$= (9.725 \cdot 10^7 + 0.3016 \cdot 10^7)$$

Angleichung der Exponenten

$$= (9.725 + 0.3016) \cdot 10^7$$

Distributivgesetz

$$= (10.0266) \cdot 10^7$$

Addition der Mantissen

$$= 1.00266 \cdot 10^8$$

Normalisierung

$$= 1.003 \cdot 10^8$$

Runden bei fester Stellenzahl

- ▶ normalerweise nicht informationstreu (!)

## Achtung: Auslöschung

Probleme bei Subtraktion zweier Gleitkommazahlen:

- ▶ Fall 1: Exponenten stark unterschiedlich
- ▶ kleinere Zahl wird soweit skaliert, dass von der Mantisse (fast) keine gültigen Bits übrigbleiben
- ▶ kleinere Zahl geht verloren, bzw. Ergebnis ist stark ungenau
- ▶ Beispiel:  $(1.0E20 + 3.14159) = 1.0E20$
  
- ▶ Fall 2: Exponenten und Mantisse fast gleich
- ▶ fast alle Bits der Mantisse löschen sich aus
- ▶ Resultat hat nur noch wenige Bits effektiver Genauigkeit

## Gleitkomma: Multiplikation, Division

Multiplikation von Gleitkommazahlen  $y = a_1 \cdot a_2$

- ▶ Multiplikation der Mantissen und Vorzeichen
- ▶ Addition der Exponenten
- ▶ ggf. Normalisierung des Resultats

$$y = (s_1 \cdot s_2) \cdot (m_1 \cdot m_2) \cdot b^{e_1 + e_2}$$

Division entsprechend:

- ▶ Division der Mantissen und Vorzeichen
- ▶ Subtraktion der Exponenten
- ▶ ggf. Normalisierung des Resultats

$$y = (s_1/s_2) \cdot (m_1/m_2) \cdot b^{e_1 - e_2}$$

## IEEE-754: Infinity, Not-a-Number

- ▶ schnelle Verarbeitung großer Datenmengen
- ▶ ungerne Statusabfrage nach jeder einzelnen Operation
- ▶ trotzdem Hinweis auf aufgetretene Probleme wichtig
  
- ▶ Inf (infinity): spezieller Wert für plus/minus Unendlich
- ▶ Beispiele:  $2/0$ ,  $-3/0$ ,  $\arctan(\pi)$ , usw.
  
- ▶ NaN (not-a-number): spezieller Wert für ungültige Operation
- ▶ Beispiele:  $\sqrt{-1}$ ,  $\arcsin(2.0)$ , Inf/Inf, usw.

# IEEE-754: Inf, NaN, $\pm 0$

Normalized	$\pm$	$0 < \text{Exp} < \text{Max}$	Any bit pattern
Denormalized	$\pm$	0	Any nonzero bit pattern
Zero	$\pm$	0	0
Infinity	$\pm$	1 1 1...1	0
Not a number	$\pm$	1 1 1...1	Any nonzero bit pattern

↖ Sign bit

- ▶ Rechnen mit Inf funktioniert normal:  $0/\text{Inf} = 0$
- ▶ NaN für undefinierte Werte:  $\text{sqrt}(-1)$ ,  $\text{arcsin}(2.0)$ , ...
- ▶ jede Operation mit NaN liefert wieder NaN

## IEEE-754: FloatInfNaNDemo.java

```

java FloatInfNaNDemo
  0 / 0 = NaN
  1 / 0 = Infinity
 -1 / 0 = -Infinity
  1 / Infinity = 0.0
Infinity + Infinity = Infinity
Infinity + -Infinity = NaN
Infinity * -Infinity = -Infinity
Infinity + NaN = NaN
sqrt(2) = 1.4142135623730951
sqrt(-1) = NaN
0 + NaN = NaN
NaN == NaN? false           Achtung (!)
Infinity > NaN? false       Achtung (!)
    
```

## ULP: Unit in the last place

- ▶ die Differenz zwischen den beiden Gleitkommazahlen, die einer gegebenen Zahl am nächsten liegen
- ▶ diese beiden Werte unterscheiden sich im niederwertigsten Bit der Mantisse
- ▶ daher auch das Maß für die erreichbare Genauigkeit
- ▶ IEEE-754 fordert eine Genauigkeit von 0.5 ULP für die elementaren Operationen (Addition, Subtraktion, Multiplikation, Division, Quadratwurzel)
- ▶ in anderen Worten: das Resultat ist beste darstellbare Wert
- ▶ gute Mathematik-Software garantiert 1 ULP auch für höhere Funktionen (Logarithmus, Sinus, Cosinus usw.)
- ▶ Java: `java.lang.Math.ulp( double d )`



# Rundungsfehler

- ▶ sorgfältige Behandlung von Rundungsfehlern essentiell
- ▶ teilweise Berechnung mit zusätzlichen Schutzstellen
- ▶ dadurch Genauigkeit  $\pm 1$  ULP für alle Funktionen
- ▶ ziemlich komplexe Sache
  
- ▶ in dieser Vorlesung nicht weiter vertieft
- ▶ beim Einsatz von numerischen Algorithmen unbedingt anschauen

## Datentypen in der Praxis: Maschinenworte

- ▶ die meisten Rechner sind für eine Wortlänge optimiert
- ▶ 8-bit, 16-bit, 32-bit, 64-bit, ... Maschinen
- ▶ die jeweils typische Länge eines Integerwertes
- ▶ und meistens auch von Speicheradressen
- ▶ zusätzlich Teile oder Vielfache der Wortlänge unterstützt
  
- ▶ derzeit dominieren noch 32-bit Rechner:
  - ▶ Wortlänge für Integerwerte ist 32-bit
  - ▶ adressierbarer Speicher ist  $2^{32}$  Bytes (4 GB)
  - ▶ bereits zu knapp für speicherhungrige Applikationen
- ▶ sukzessive Übergang zu 64-bit Rechnern

## Datentypen auf Maschinenebene

- ▶ gängige Prozessoren unterstützen mehrere Datentypen
- ▶ entsprechend der elementaren Datentypen in C, Java, ...
- ▶ `void*` ist ein **Pointer** (Referenz, Speicheradresse)
- ▶ Beispiel für die Anzahl der Bytes:

C Datentyp	DEC Alpha	typ. 32-bit	Intel IA-32 (x86)
<code>int</code>	4	4	4
<code>long int</code>	8	4	4
<code>char</code>	1	1	1
<code>short</code>	2	2	2
<code>float</code>	4	4	4
<code>double</code>	8	8	8
<code>long double</code>	8	8	10/12
<code>void *</code>	8	4	4



## Literatur: Vertiefung

- ▶ Klaus von der Heide, *Interaktives Skript, Technische Informatik 1*, Universität Hamburg, 2005  
[tams-www.informatik.uni-hamburg.de/lectures/2004ws/vorlesung/t1/Float/Double-Demonstration: demoieeee754](http://tams-www.informatik.uni-hamburg.de/lectures/2004ws/vorlesung/t1/Float/Double-Demonstration:demoieeee754)
- ▶ Donald E. Knuth, *The Art of Computer Programming: Volume 4 Fascicle 0: Boolean Functions*  
*Volume 4 Fascicle 1: Bitwise Tricks and Techniques, Binary Decision Diagrams* Addison-Wesley, 2006-2009
- ▶ David Goldberg, *What every computer scientist should know about floating-point*, 1991  
[www.validlab.com/goldberg/paper.pdf](http://www.validlab.com/goldberg/paper.pdf)
- ▶ Georges Ifrah, *Universalgeschichte der Zahlen*, 2008