Speicherung und Interpretation von Information

Zahlensysteme
Römische Zahlen

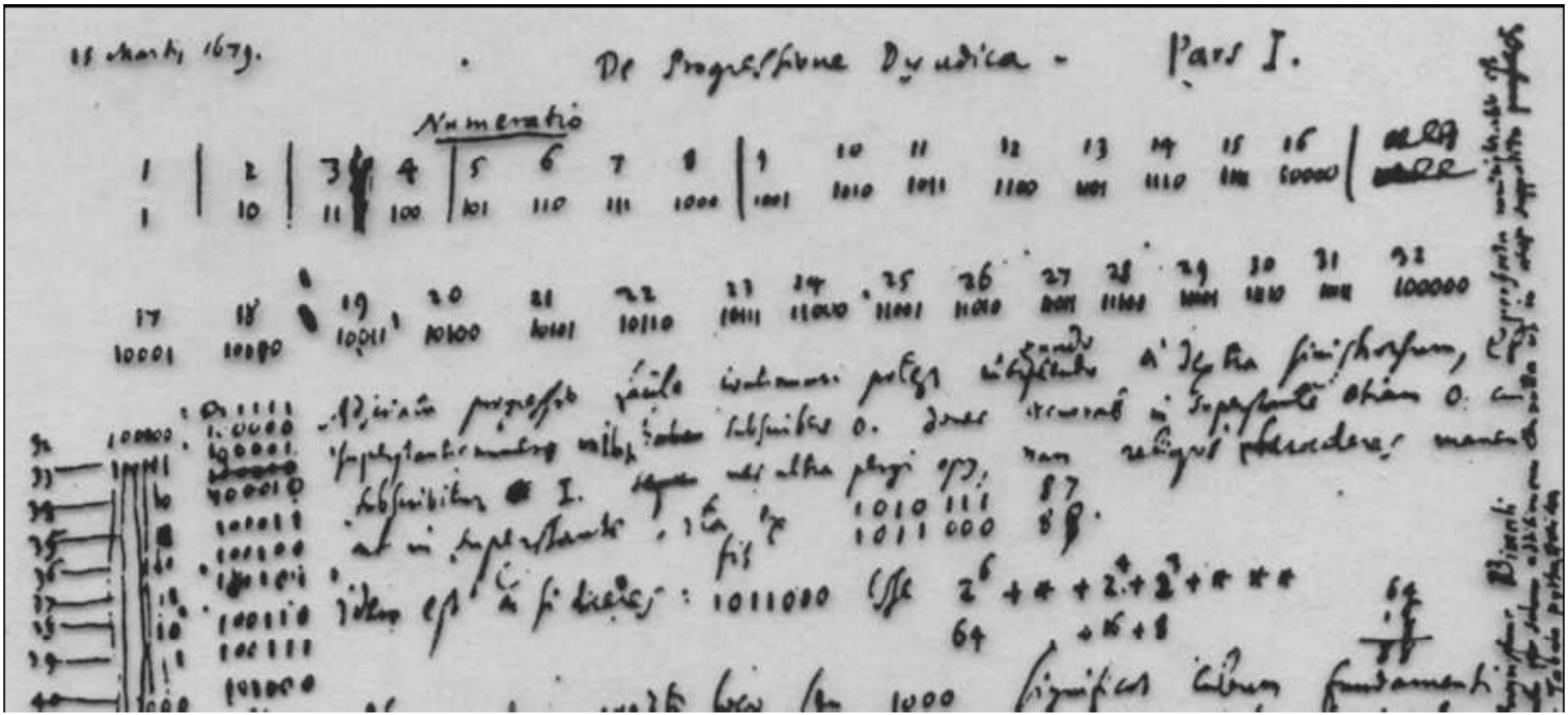
I	1	X	10	C	100
II	2	XX	20	CC	200
III	3	XXX	30	CCC	300
IV	4	XL	40	CD	400
V	5	L	50	D	500
VI	6	LX	60	DC	600
VII	7	LXX	70	DCC	700
VIII	8	LXXX	80	DCCC	800
IX	9	XC	90	CM	900
				M	1000

Römische Zahlen

- Ziffern: I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000
- Ziffern I, X, C, M dürfen beliebig oft nebeneinander stehen
- Ziffern V, L, D dürfen nicht wiederholt nebeneinander stehen
- Stehen gleiche Ziffern nebeneinander, so werden diese addiert:
  II = 1 + 1 = 2, XXX = 10 + 10 + 10 = 30, CC = 100 + 100 = 200,
  MMM = 1000 + 1000 + 1000 = 3000
- Ziffern I, X und C dürfen max. dreimal nebeneinander stehen
- Ziffer M kann beliebig oft nebeneinander stehen
- Ziffern V, L und D dürfen in einer Zahl nur einmal vorkommen
- Steht kleinere Einheit **rechts** neben größerer Einheit, dann wird die kleinere Einheit zur größeren Einheit addiert: VI = 5 + 1 = 6,
  XIII = 10 + 1 + 1 + 1 = 13, DCCLVI = 500 + 100 + 100 + 50 + 5 + 1 = 756
- Steht eine kleinere Einheit **links** neben dem Zeichen einer größeren Einheit, dann wird die kleinere von der größeren Einheit subtrahiert:
  IV = 5 – 1 = 4 , IX = 10 – 1 = 9 , XXIX = 10 + 10 + 10 – 1
- Es darf max. nur eine kleinere Einheit von der größeren Einheit subtrahiert werden. Mehrere Schreibweisen für einen Zahlenwert können auftreten

Zahlensysteme

Leibniz-Traktat bezüglich Dualzahlen von 1679



Zahlensysteme
Positionssysteme bei natürlichen Zahlen

Ein Positionssystem mit der Basis B ist ein Zahlensystem, in dem eine Zahl x nach Potenzen von B zerlegt wird.

Eine natürliche Zahl n wird durch folgende Summe dargestellt:

$$n = \sum_{i=0}^{N-1} b_i \cdot B^i \qquad \text{wobei Folgendes gilt:}$$

- $\mathtt{B} = \text{Basis des Zahlensystems } (B \in \mathbb{N}, B \geq 2)$
- $\mathtt{b} = \text{Ziffern } (b_i \in \mathbb{N}_0, 0 \leq b_i < B)$
- $\mathtt{N} = \text{Anzahl der Stellen}$

Zahlensysteme

Zahlendarstellung in verschiedenen Zahlensystemen

Dual	Oktal	Dezimal	Hexadezimal	Zwölfersystem
0	0	**0**	0	0
1	1	**1**	1	1
10	2	**2**	2	2
11	3	**3**	3	3
100	4	**4**	4	4
101	5	**5**	5	5
110	6	**6**	6	6
111	7	**7**	7	7
1000	10	**8**	8	8
1001	11	**9**	9	9
1010	12	**10**	a	a
1011	13	**11**	b	b
1100	14	**12**	c	10
1101	15	**13**	d	11
1110	16	**14**	e	12
1111	17	**15**	f	13
10000	20	**16**	10	14
10001	21	**17**	11	15

Zahlensysteme
Positionssysteme bei gebrochenen Zahlen

Bei gebrochenen Zahlen trennt ein Punkt (Komma im Deutschen) in der Zahl den ganzzahligen Teil der Zahl vom gebrochenen Teil (Nachkommateil). Solche Zahlen lassen sich durch folgende Summenformel beschreiben:

$$n = \sum_{i=-M}^{N-1} b_i \cdot B^i \qquad \text{wobei Folgendes gilt:}$$

- B = Basis des Zahlensystems ($B \in \mathbb{N}, B \geq 2$)
- b = Ziffern ($b_i \in \mathbb{N}_0, 0 \leq b_i < B$)
- N = Anzahl der Stellen vor dem Punkt (Komma)
- M = Anzahl der Stellen nach dem Punkt (Komma)

Zahlensysteme
Positionssysteme bei gebrochenen Zahlen

- Beispiele:

$$
\begin{aligned}
(17.05)_{10} &= 1 \cdot 10^{1} + 7 \cdot 10^{0} + 0 \cdot 10^{-1} + 5 \cdot 10^{-2} \\
(3758.0)_{10} &= 3 \cdot 10^{3} + 7 \cdot 10^{2} + 5 \cdot 10^{1} + 8 \cdot 10^{0} \\
(9.702)_{10} &= 9 \cdot 10^{0} + 7 \cdot 10^{-1} + 0 \cdot 10^{-2} + 2 \cdot 10^{-3} \\
(0.503)_{10} &= 0 \cdot 10^{0} + 5 \cdot 10^{-1} + 0 \cdot 10^{-2} + 3 \cdot 10^{-3}
\end{aligned}
$$

Dual-, Oktal- und Hexadezimalsystem

In der Informatik spielen das Dual-, Oktal- und Hexadezimalsystem eine zentrale Rolle.

- **Das Dualsystem und das Bit im Rechner**
  - Das von uns verwendete Zehnersystem ist ein Positionssystem. Jeder Position in einer Zahl ist ein bestimmter Wert zugeordnet, der eine Potenz von 10 ist.
  - Das Zehnersystem, in dem 10 verschiedene Ziffern 0, 1, 2,…, 9 existieren, ist technisch schwer zu realisieren. Daher benutzt man in Rechnern intern das Dualsystem, bei dem nur zwei Ziffern, 0 und 1, verwendet werden, die sich technisch relativ leicht nachbilden lassen:

$$0 = \text{kein Strom, keine Spannung}$$

$$1 = \text{Strom, Spannung}$$

Dual-, Oktal- und Hexadezimalsystem

- Eine einzelne Binärstelle (0 oder 1), die ein Rechner speichert,wird als **Bit** bezeichnet. Das ist die Abkürzung für „*BInary digiT*“, also Binärziffer. Es handelt sich um die kleinste Informationseinheit, die ein Computer verarbeiten kann.
- Auch beim Dualsystem handelt es sich um ein Positionssystem. Der Wert einer Position ist hier jedoch eine Potenz von 2:

$$10011 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$
$$= 19 \text{ (im Zehnersystem)}$$

- Beispiele:

$(10011)_2$	=	$(19)_{10}$	oder:	$10011_{(2)}$	=	$19_{(10)}$
$(1011)_2$	=	$(11)_{10}$	oder:	$1011_{2)}$	=	$11_{(10}$
$(11010110)_2$	=	$(214)_{10}$	oder:	$11010110_{(2)}$	=	$214_{(10)}$

Dual-, Oktal- und Hexadezimalsystem

– **Konvertieren zwischen Dual- und Oktalsystem**
  - Neben dem Dualsystem ist in der Informatik noch das Oktalsystem wichtig, da es in einer engen Beziehung zum Dualsystem steht. Es gilt nämlich: $2^3 = 8$ (Basis des Oktalsystems).
  - Um eine im Dualsystem dargestellte Zahl ins Oktalsystem zu konvertieren, bildet man von rechts beginnend so genannte *Dualtriaden* (Dreiergruppen).

110	111	001	110	010	Dualzahl
6	7	1	6	2	Oktalzahl

Dual-, Oktal- und Hexadezimalsystem

– **Konvertieren zwischen Dual- und Oktalsystem**

- Bei der Umwandlung einer Oktalzahl in ihre Dualdarstellung geht man den umgekehrten Weg.

3	6	1	4	Oktalzahl
011	110	001	100	Dualzahl

- Es ist offensichtlich, dass ein Mensch sich die Zahl $3614_{(8)}$ wesentlich leichter merken und damit umgehen kann, als $011110001100_{(2)}$.

Dual-, Oktal- und Hexadezimalsystem

- **Konvertieren zwischen Dual- und Hexadezimalsystem**
  - Neben dem Dualsystem ist in der Informatik noch das Hexadezimalsystem wichtig, da es ebenfalls in einer engen Beziehung zum Dualsystem steht. Es gilt nämlich: $2^4 = 16$.
  - Um eine im Dualsystem dargestellte Zahl ins Hexadezimalsystem zu konvertieren, bildet man von rechts beginnend so genannte *Dualtetraden* (Vierergruppen).

$(101110101101)_2 = (175D)_{16} = (13535)_8$

Vierergruppen: 1 0111 0101 1101	Dreier: 1 011 101 011 101
Hexadezimal: 1 7 5 D	Oktal: 1 3 5 3 5

$(ADA)_{16} = (101011011010)_2 = (5332)_8$

Hexadezimal: A D A	Dreiergruppen: 101 011 011 010
Vierergruppen: 1010 1101 1010	Oktal: 5 3 3 2

Konvertierungsalgorithmen: Konvertieren von anderen Systemen in das Dezimalsystem

Eine in einem Positionssystem mit der Basis B dargestellte natürliche Zahl n:

$$n = \sum_{i=0}^{N} b_i \cdot B^i$$

lässt sich mit Hilfe des Hornerschemas wie folgt darstellen:

$$n = (\ldots(((b_N \cdot B + b_{N-1}) \cdot B + b_{N-2}) \cdot B + b_{N-3}) \cdot B + \ldots + b_1) \cdot B + b_0$$

Mit Hilfe dieser Darstellung können Konvertierungen in das Dezimalsystem einfach durchgeführt werden.

$$\begin{aligned} (1578)_{10} &= ((1 \cdot 10 + 5) \cdot 10 + 7) \cdot 10 + 8 \\ (754)_8 &= (7 \cdot 8 + 5) \cdot 8 + 4 = (492)_{10} \end{aligned}$$

Konvertieren vom Dezimalsystem in andere Systeme

Für die Umwandlung einer Dezimalzahl x in ein Zahlensystem mit der Basis n kann folgender Algorithmus verwendet werden:

1. `x : n = y Rest z`
2. Mache y zum neuen x und fahre wieder mit Schritt 1 fort, wenn dieses neue x ungleich 0 ist, ansonsten fahre mit Schritt 3 fort.
3. Die ermittelten Reste z von unten nach oben nebeneinander geschrieben ergeben dann die entsprechende Dualzahl.

Konvertieren vom Dezimalsystem in andere Systeme

Beispiele:

$(30)_{10} = ?_2$

x				y		z
30	:	2	=	15	Rest	0
15	:	2	=	7	Rest	1
7	:	2	=	3	Rest	1
3	:	2	=	1	Rest	1
1	:	2	=	0	Rest	1

$(43)_{10} = ?_2$

x				y		z
43	:	2	=	21	Rest	1
21	:	2	=	10	Rest	1
10	:	2	=	5	Rest	0
5	:	2	=	2	Rest	1
2	:	2	=	1	Rest	0
1	:	2	=	0	Rest	1

Die Reste z von unten nach oben nebeneinander geschrieben liefern dann die gesuchte Dualzahl: $(30)_{10} = 11110_2$ $(43)_{10} = (101011)_2$

Konvertieren echt gebrochener Zahlen in das Dezimalsystem

Eine echt gebrochene Zahl n (n < 1):

$$n = \sum_{i=-M}^{-1} b_i \cdot B^i$$

lässt sich mit Hilfe des Hornerschemas wie folgt darstellen:

$$n = \frac{1}{B} \cdot \left(b_{-1} + \frac{1}{B} \cdot \left(b_{-2} + \frac{1}{B} \cdot \left(b_{-3} + \ldots + \frac{1}{B} \cdot \left(b_{-M+1} + \frac{1}{B} \cdot b_{-M} \right) \ldots \right) \right) \right)$$

Mit Hilfe dieser Darstellung können wieder Konvertierungen von anderen Systemen in das Dezimalsystem einfach durchgeführt werden.

z. B. die Zahl: $0.193_{(10)} = \frac{1}{10} \cdot \left(1 + \frac{1}{10} \cdot \left(9 + \frac{1}{10} \cdot 3 \right) \right)$

Konvertieren echt gebrochener Zahlen vom Dezimalsystem in andere Systeme

Für die Umwandlung des Nachkommateils einer Dezimalzahl in ein anderes Positionssystem existiert folgender Algorithmus, wobei B die Basis des Zielsystems ist:

1. $x \cdot B = y$ Überlauf z (z = ganzzahliger Anteil)
2. Mache Nachkommateil von y zum neuen x und fahre mit Schritt 1 fort, wenn dieses neue x ungleich 0 ist und noch nicht genügend Nachkommastellen ermittelt sind, ansonsten fahre mit Schritt 3 fort.
3. Schreibe die ermittelten Überläufe von oben nach unten nach 0. nebeneinander, um die entsprechende Dualzahl zu erhalten.

Konvertieren echt gebrochener Zahlen vom Dezimalsystem in andere Systeme

Beispiele:

$(0.408203125)_{10} = (0.321)_8$

x		y		z
0.408203125	$\cdot 8 =$	3.265625	Ueberl.	3
0.265625	$\cdot 8 =$	2.125	Ueberl.	2
0.125	$\cdot 8 =$	1	Ueberl.	1
0	$\cdot 8 =$	0	Ueberl.	0

$(0.34375)_{10} = (0.01011)_2$

x		y		z
0.34375	$\cdot 2 =$	0.6875	Ueberl.	0
0.6875	$\cdot 2 =$	1.375	Ueberl.	1
0.375	$\cdot 2 =$	0.75	Ueberl.	0
0.75	$\cdot 2 =$	1.5	Ueberl.	1
0.5	$\cdot 2 =$	1.0	Ueberl.	1
0	$\cdot 2 =$	0.0	Ueberl.	0

Die Überläufe z von oben nach unten nach 0. nebeneinander geschrieben liefern dann die gesuchte Zahl.

Genauigkeitsverluste bei der Umwandlung gebrochener Dezimalzahlen

Manche gebrochenen Zahlen, die sich ganz genau im Dezimalsystem darstellen lassen, lassen sich leider nicht ganz genau als Dualzahl darstellen.

Typische Beispiele dafür sind Zahlen, die sich im Dualsystem nur durch eine periodische Ziffernfolge repräsentieren lassen, wie z. B.
0.1(10) = 0.0001 1001 1001 1…(2):

x	y	z
$0.1 * 2 = 0.2$	Überlauf	0
$0.2 * 2 = 0.4$	Überlauf	0
$0.4 * 2 = 0.8$	Überlauf	0
$0.8 * 2 = 1.6$	Überlauf	1
$0.6 * 2 = 1.2$	Überlauf	1
$0.2 * 2 = 0.4$	Überlauf	0
$0.4 * 2 = 0.8$	Überlauf	0
$0.8 * 2 = 1.6$	Überlauf	1
$0.6 * 2 = 1.2$	Überlauf	1

Solche Ungenauigkeiten treten dann natürlich auch in den Rechnern auf, die ja mit dem Dualsystem arbeiten.

Im Beispiel wiederholt sich das Bitmuster 0011und es gilt: $0.1_{(10)} = 0.0\ \ 0011\ \ 0011 \ldots_{(2)}$

Konvertieren unecht gebrochener Zahlen

Um eine unecht gebrochene Zahl zu konvertieren, muss diese in ihren ganzzahligen Teil und ihren echt gebrochenen Teil aufgeteilt werden, die dann getrennt von einander zu konvertieren sind.

$$(12.25)_{10} = (1100.01)_2$$

Ganzzahliger Teil: $(12)_{10} = (1100)_2$	**Echt gebrochener Teil:** $(0.25)_{10} = (0.01)_2$
$12 : 2 = 6$ Rest 0	$0.25 * 2 = 0.5$ Überlauf 0
$6 : 2 = 3$ Rest 0	$0.5 * 2 = 1$ Überlauf 1
$3 : 2 = 1$ Rest 1	$0 * 2 = 0$ Überlauf 0
$1 : 2 = 0$ Rest 1	

Rechenoperationen im Dualsystem

Addition von Dualzahlen

Für die duale Addition gilt allgemein:

$$
\begin{aligned}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 0 \text{ Übertrag } 1 \\
1 + 1 + 1 \text{ (vom Übertrag)} &= 1 \text{ Übertrag } 1
\end{aligned}
$$

$$
\begin{array}{rl}
0\,1\,0\,1\,1\,0\,1 = & 45 \\
0\,1\,1\,0\,1\,1\,0 = & 54 \\
\hline
1\,1\,1\,1 \quad = & \text{Übertrag} \\
\hline
1\,1\,0\,0\,0\,1\,1 = & 99
\end{array}
$$

Rechenoperationen im Dualsystem
Subtraktion u. negative Zahlen

Negative Zahlen werden üblicherweise durch ihren Betrag mit vorangestelltem Minuszeichen dargestellt.

- Diese Darstellung wäre auch rechnerintern denkbar, hat jedoch den Nachteil, dass man eine gesonderte Vorzeichenrechnung durchführen müsste und man ein Rechenwerk benötigt, das sowohl addieren als auch subtrahieren kann.
- Man kann die Subtraktion auf eine Addition zurückzuführen durch das Verfahren der *Komplementbildung*.
- Man unterscheidet zwei Arten der Komplementbildung, wobei B für das Zahlensystem steht:
  B-Komplement und *(B-1)-Komplement*
- Da das B-Komplement technisch leichter realisierbar ist, wird vorwiegend mit dem B-Komplement (Zweier-Komplement) gearbeitet.

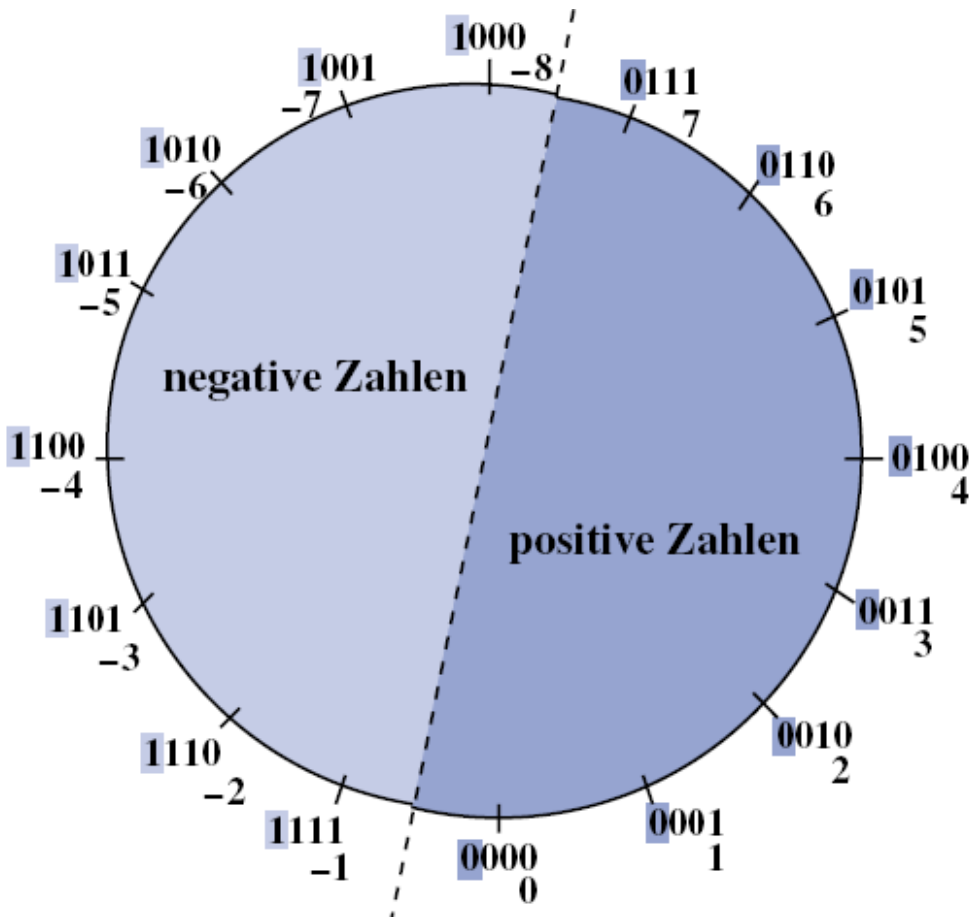
Rechenoperationen im Dualsystem
B-Komplement

Zuordnung der Bitkombinationen zu positiven und negativen Zahlen

Zahlenring für vier Bits, erstes Bit ist Vorzeichenbit

Alle Kombinationen, bei denen das 1. Bit (Vorzeichenbit) gesetzt ist, repräsentieren dabei negative Zahlen:

0000 = 0	1000 = −8
0001 = 1	1001 = −7
0010 = 2	1010 = −6
0011 = 3	1011 = −5
0100 = 4	1100 = −4
0101 = 5	1101 = −3
0110 = 6	1110 = −2
0111 = 7	1111 = −1



Rechenoperationen im Dualsystem
B-Komplement

- Regeln für die Bildung eines Zweier-Komplements
  1. Ist das 1. Bit 1, so handelt es sich um eine negative Zahl.
  2. Der Wert einer negativen Zahl wird dann im Zweier-Komplement dargestellt. Zweier-Komplement zu einem Wert bedeutet, dass zunächst jedes einzelne Bit invertiert (umgedreht) wird, und dann auf die so entstandene Bitkombination die Zahl 1 aufaddiert wird.

Zweier-Komplement zu 5:

Dualdarstellung von 5: 0101

Negieren von 5:	1010
+ 1:	0001
= −5:	1011

Zweier-Komplement zu -5:

Dualdarstellung von −5: 1011

Negieren von −5:	0100
+ 1:	0001
= 5:	0101

Rechenoperationen im Dualsystem
B-Komplement

- Bei der Verwendung der Komplement-Darstellung muss eine Maschine nicht subtrahieren können, sondern kann jede Subtraktion a - b durch eine Addition a + -b realisieren.

2 - 4 = 2 + -4

$$\begin{array}{rrr} & 0010 & = \quad 2 \\ + & 1100 & = -4 \\ \hline & 1110 & = -2 \end{array}$$

6 - 2 = 6 + -2

$$\begin{array}{rrr} & 0110 & = \quad 6 \\ + & 1110 & = -2 \\ \hline & \mathbf{1}|0100 & = \quad 4 \end{array}$$

Das vorne überlaufende Bit wird weggeworfen

Rechenoperationen im Dualsystem
B-Komplement

Im Beispiel hat der vorne stattfindende Überlauf des Bits keinen Einfluss auf die Richtigkeit des Ergebnisses.
Das gilt nicht allgemein. Wenn das Ergebnis nicht im darstellbaren Zahlenbereich liegt, dann erhält man bei einem Überlauf ein falsches Ergebnis:

Bei fünf verfügbaren Stellen soll die Subtraktion $(-9)_{10} - (13)_{10}$ im Dualsystem mit Hilfe des B-Komplements durchgeführt werden.

Darstellbarer Zahlenbereich: $-2^4..2^4 - 1 = -16.. + 15$

$$\begin{array}{rl} -(9)_{10}: & (10111)_2 \\ +\ (-13)_{10}: & (10011)_2 \\ \hline (+10)_{10}: & 1|\ (01010)_2 \end{array}$$

Das vorne überlaufende Bit geht verloren → falsches Ergebnis

Rechenoperationen im Dualsystem
(B-1)-Komplement

(B-1)-Komplement (Einer-Komplement) zum Vergleich.

Die Zahlendarstellung ist hier *symmetrisch*. Es gibt eine positive und eine negative 0.

0000	+0	(positive Null)
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	−7	
1001	−6	
1010	−5	
1011	−4	Alle Kombinationen, bei denen das 1. Bit (Vorzeichenbit)
1100	−3	gesetzt ist, repräsentieren dabei negative Zahlen.
1101	−2	
1110	−1	
1111	−0	(negative Null)

Rechenoperationen im Dualsystem
(B-1)-Komplement

- Regeln für die Bildung eines Einer-Komplements
  - Ist das 1. Bit mit 1 besetzt, so handelt es sich um eine negative Zahl (eventuell die negative Null 111…111).
  - Der Wert einer negativen Zahl wird dann im Einer-Komplement dargestellt. Einer-Komplement zu einem Wert bedeutet, dass zunächst jedes einzelne Bit invertiert (umgedreht) wird.
  - Führt die Addition des Komplements zum Überlauf einer 1, muss zum Ergebnis diese 1 hinzuaddiert werden („*Einer-Rücklauf* “).

Einer-Komplement zu $(7)_{10} = (00111)_2$:
Invertieren von 7 (−7): 11000
Dualdarstellung von 14 : 01110

$(14)_{10} - (7)_{10}$ + −7 : 11000

: 1|00110
Aufaddieren von 1 : 00001

= 7 : 00111

Einer-Komplement zu $(13)_{10} = (01101)_2$:
Invertieren von 13 (−13): 10010
Dualdarstellung von 9: 01001

$(9)_{10} - (13)_{10}$ + −13: 10010

= −4: 11011

Rechenoperationen im Dualsystem
Multiplikation und Division

- Die ganzzahlige Multiplikation bzw. Division wird in einem Rechner allgemein mittels wiederholter Addition durchgeführt.

- In den Sonderfällen des Multiplikators bzw. Divisors von 2, 4, 8, … kann die Multiplikation bzw. Division aber einfacher und schneller durch eine Verschiebung von entsprechend vielen Bits nach links bzw. rechts erfolgen:
  Bei 2 (2^1) um 1 Bit, bei 4 (2^2) um 2 Bits, bei 8 um 3 (2^3) Bits usw.

$$\begin{aligned} \text{dezimal} &: (20)_{10} \times (8)_{10} = 160_{10} \\ \text{dual} &: (10100)_2 \times (1000)_2 = (10100000)_2 \ [10100 \mid 000] \end{aligned}$$

$$\begin{aligned} \text{dezimal} &: (20)_{10} : (4)_{10} = 5_{10} \\ \text{dual} &: (10100)_2 : (100)_2 = (101)_2 \ [101 \mid 00] \end{aligned}$$

Rechenoperationen im Dualsystem

Konvertieren durch sukzessive Multiplikation und Addition

- Für die Konvertierung aus einem beliebigen Positionssystem in ein anderes Positionssystem kann auch der folgende Algorithmus verwendet werden, wobei die Berechnung im Zielsystem mit der Basis B des Ausgangssystems durchgeführt wird:

$$
\begin{array}{lcl}
b_n \cdot B & = & a_1 \\
a_1 + b_{n-1} & = & a_2 \\
a_2 \cdot B & = & a_3 \\
a_3 + b_{n-2} & = & a_4 \\
\ldots & \ldots & \ldots \\
a_{2n-1} + b_0 & = & \mathrm{x}
\end{array}
$$

im Zielsystem
(mit der Basis B)

Konvertieren der Zahl $(2314)_{10}$ in das Dualsystem

$2, 3, 1, 4$ $(10)_2, (11)_2, (1)_2, (100)_2$

$B = (1010)_2$

$$
\begin{array}{rcl}
10 \cdot 1010 & = & 10100 \\
+\ 11 & = & 10111 \\
\cdot\ 1010 & = & 11100110 \\
+\ 1 & = & 11100111 \\
\cdot\ 1010 & = & 10010000110 \\
+\ 100 & = & \mathbf{100100001010}
\end{array}
$$

Reelle Zahlen
Festpunktzahlen

Bei Festpunktzahlen steht der Punkt (das Komma) immer an einer bestimmten festgelegten Stelle, wobei der Punkt natürlich nicht eigens mitgespeichert wird:

$$\text{zahl} = z_{n-1}z_{n-2}\ldots z_1 z_0 \qquad z_{-1}z_{-2}\ldots z_{-m(2)} \;;$$

$$\text{zahl} = \sum_{i=-m}^{n-1} z_i 2^i$$

zahl hat die Länge n + m, wobei n Stellen vor und m Stellen nach dem Punkt gesetzt sind.

$$\begin{aligned}(11.011)_2 &= 1\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} \\ &= 2 + 1 + 0\cdot 0.5 + 0.25 + 0.125 = (3.375)_{10}\end{aligned}$$

Reelle Zahlen
Festpunktzahlen

Durch Einführen eines eigenen Vorzeichenbits können dann noch positive und negative Zahlen unterschieden werden.
Die Nachteile der Festpunktdarstellung sind:

1. Man kann mit einer bestimmten Anzahl von Bits nur einen beschränkten Wertebereich abdecken.
2. Die Stelle des Punkts (Kommas) muss allgemein festgelegt werden. Und wo soll man diesen festlegen, wenn manchmal mit sehr kleinen, hochgenauen Werten und ein anderes Mal mit sehr großen Werten gearbeitet werden muss?

Aufgrund dieser Nachteile wird die Festpunktdarstellung nur in Rechnern verwendet, die für Spezialanwendungen benötigt werden. In den üblichen heute verbreiteten Rechnern wird stattdessen die Gleitpunktdarstellung verwendet.

Reelle Zahlen

IEEE-Format für float und double

Jede reelle Zahl kann in der Form $2.3756 \cdot 10^3$ angegeben werden. Bei dieser Darstellungsform setzt sich die Zahl aus zwei Bestandteilen zusammen:

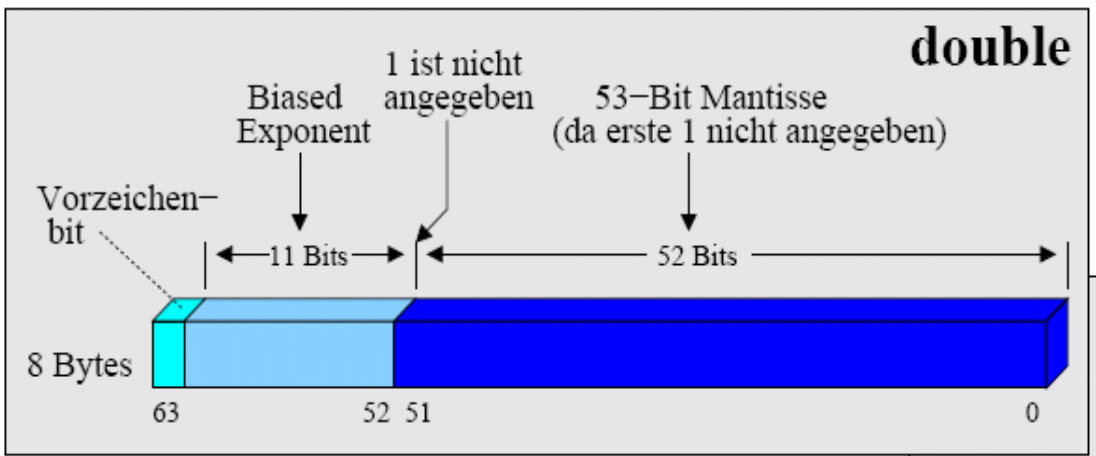
- Mantisse (2.3756) und Exponent (3), der ganzzahlig ist.

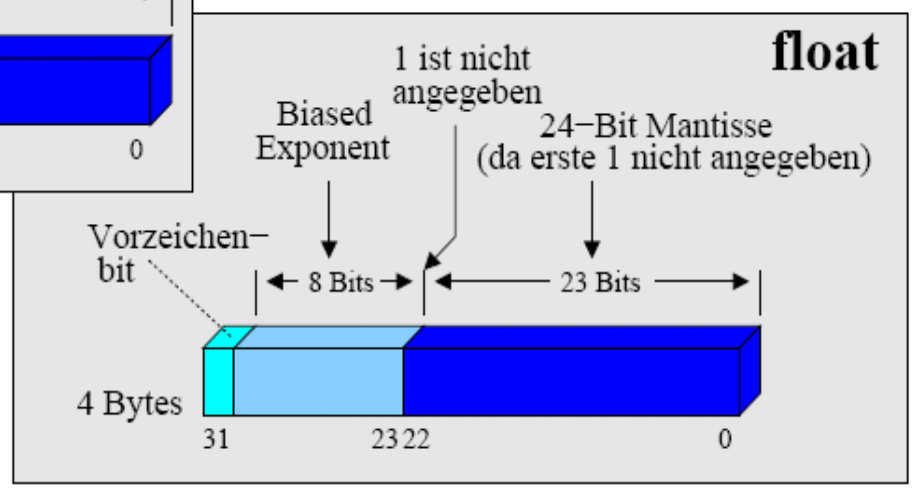
Diese Form wird auch meist in Rechnern verwendet, außer dass dort nicht mit Basis 10, sondern mit Basis 2 gearbeitet wird.
Die für die Darstellung einer Gleitpunktzahl verwendete Anzahl von Bytes legt fest, ob man mit *einfacher* (Datentyp float) oder mit *doppelter Genauigkeit* (Datentyp double) arbeitet.

Reelle Zahlen

IEEE-Format für float und double

Zur Darstellung verwenden die C/C++- und Java-Datentypen **`float`** und **`double`** das standardisierte IEEE-Format, wobei vier Bytes für **`float`** und acht Bytes für **`double`** definiert sind.





Reelle Zahlen

IEEE-Format für float und double

Das IEEE-Format geht von *normalisierten Gleitpunktzahlen* aus.

„Normalisierung“ bedeutet, dass der Exponent so verändert wird, dass der gedachte Dezimalpunkt immer rechts von der ersten Nicht-Null-Ziffer liegt (im Binärsystem ist dies immer eine 1).

Die Dezimalzahl

$$17.625 = 1 \cdot 10^1 + 7 \cdot 10^0 + 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$$

entspricht der binären Zahl:

$$16 + 1 + 1/2 + 1/8$$

$$= 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10001.101 \cdot 2^0$$

Die entsprechende normalisierte Form erhält man, indem man den Dezimalpunkt hinter die erste signifikante Ziffer „schiebt“ und den Exponenten entsprechend anpasst:

$$1.0001101 \cdot 2^4$$

Reelle Zahlen

IEEE-Format für float und double

In der *Mantisse* steht durch die normalisierte Form das höchstwertige „Einser-Bit“ immer links vom gedachten Dezimalpunkt (außer für 0.0). Beim IEEE-Format wird dieses Bit nicht gespeichert.

Der *Exponent* ist eine Ganzzahl, welche (nach Addition eines *bias*) ohne Vorzeichen dargestellt wird. Durch diese *bias*-Addition wird für den Exponent keine Vorzeichenrechnung benötigt.
Der Wert von *bias* hängt vom Genauigkeitsgrad ab:
- float (mit 4 Bytes, 8 Bits für Exponent): bias=127
- double (mit 8 Bytes, 11 Bits für Exponent): bias=1023

Das *Vorzeichenbit* zeigt das Vorzeichen der Mantisse, die immer als Betragswert, auch im negativen Fall nicht als Komplement, dargestellt wird.

Reelle Zahlen

IEEE-Format für float und double

Formel zur Darstellung einer Gleitpunktzahl im IEEE-Format:

$$(-1)^S \cdot \underbrace{(2^{B-bias})}_{EXPONENT} \cdot \underbrace{(1.f^N .....f^0)}_{SIGNIFICAND}$$

- $SIGN$ S = 0 (positiv); S = 1 (negativ)
- bias = 127 (`float`), bias = 1023 (`double`)
- B = Biased Exponent (zu speichernder Exponent)
- N=22 (`float`=23 Stellen), N=51 (`double`=52 Stellen)

$17.625\,(1.0001101 \cdot 2^4)$

```
|0|10000011|00011010000000000000000|
31       /                         0
        Biased Exponent ergibt sich als:
                          bias =    0111 1111 =  127
           + wirklicher Exponent =    0000 0100 =    4
                                     ---------------
                                     1000 0011 =  131
```

Reelle Zahlen

IEEE-Format für float und double

Nach IEEE gilt für float (einfach) und double (doppelt):

	einfach	doppelt
Vorzeichen-Bits	1	1
Exponenten-Bits	8	11
Mantissen-Bits	23	52
Bits insgesamt	32	64
BIAS	127	1023
Exponentenbereich	[−126,127]	[−1022,1023]

Einige Sonderfälle des IEEE-Formats.

Biased Exponent	Mantisse	Bedeutung
111..111(= 255 bzw. = 2047)	$\neq 0$	*not a number* (keine gültige Zahl)
111..111(= 255 bzw. = 2047)	000..000(= 0)	$\pm\infty$
000..000(= 0)	000..000(= 0)	± 0