



Kap. 3: Repräsentierung von Information in Rechensystemen

3.1 Einführung und Überblick

3.2 Bitfolgen

3.3 Zahlensysteme, Zahlendarstellungen, Arithmetik

3.4 Zeichenketten

3.5 Ein-/Ausgabe



Quellen

- **U. Rembold, P. Levi: "Einführung in die Informatik für Naturwissenschaftler und Ingenieure", 3. Auflage, Hanser-Verlag, 1999 (Kap. 2.1.)**
- **D. Werner u.a.: "Taschenbuch der Informatik", Fachbuchverlag Leipzig, 1995 (Kap. 3.1)**
- **F. Mayer-Lindenberg: "Konstruktion digitaler Systeme", Vieweg-Verlag, 1998 (Kap. 2)**
- **H. Dispert, H.-G. Heuck: "Einführung in die Technische Informatik und Digitaltechnik", Vorlesungsskript FH Kiel (Kap. 8), http://www.e-technik.fh-kiel.de/universe/digital/dig0_00.htm**
- **The Unicode Consortium: "The Unicode Standard", Addison-Wesley, 1992, <http://www.unicode.org>**



3.1 Einführung und Überblick

- **Wiederholung (Kap. 2):**
Information:
 - **abstrakter Bedeutungsgehalt (Semantik)**
 - **äußere Form von Information: Repräsentation oder Darstellung.**
- **Die Hardware eines Rechensystems ist auf wenige, genau festgelegte Informationsrepräsentierungen zugeschnitten und gestattet deren Manipulation durch Maschinenbefehle.**



Einführung und Überblick (2)

Def

- ***Daten*** sind Informationen, die nach eindeutigen Regeln in Rechensystemen gespeichert und verarbeitet werden ("Datenverarbeitung").
- Auf der untersten Abstraktionsebene sind diese durch die Hardware eines Rechensystems bestimmt.
- Auf höheren Abstraktionsebenen sind Daten
 - nach algorithmischen Gesichtspunkten strukturierte Informationen (z.B. Personaldaten).
 - i.d.R. von einem Programmierer / einer Programmiererin entworfen (zusammengesetzte Datentypen)
 - durch Compiler auf die elementaren Informationsdarstellungen der Hardware abgebildet.
- ***Nachrichten*** sind Daten, die zur Kommunikation (Übertragung) von Information verwendet werden.



Einführung und Überblick (3)

- In diesem Kapitel: Besprechung der in heutigen Rechnern üblichen Repräsentierungen für die elementaren Datentypen zusammen mit ihren jeweiligen Grundlagen.
- Als elementar werden in heutigen Rechnern i.d.R. angesehen:
 - Bitfolgen
 - Zahlen in verschiedenen Darstellungen
 - Zeichenketten
- In heutigen Rechnern wird jegliche Information in binärer Form codiert dargestellt und verarbeitet
⇒ die elementaren Datentypen werden abgebildet auf Binärwörter (vgl. Kap. 2.4).
- Im folgenden Kapitel 4:
Besprechung der allgemeinen Grundlagen der Codierung (Vorgehensweise: vom Konkreten zum Abstrakten).



3.2 Bitfolgen

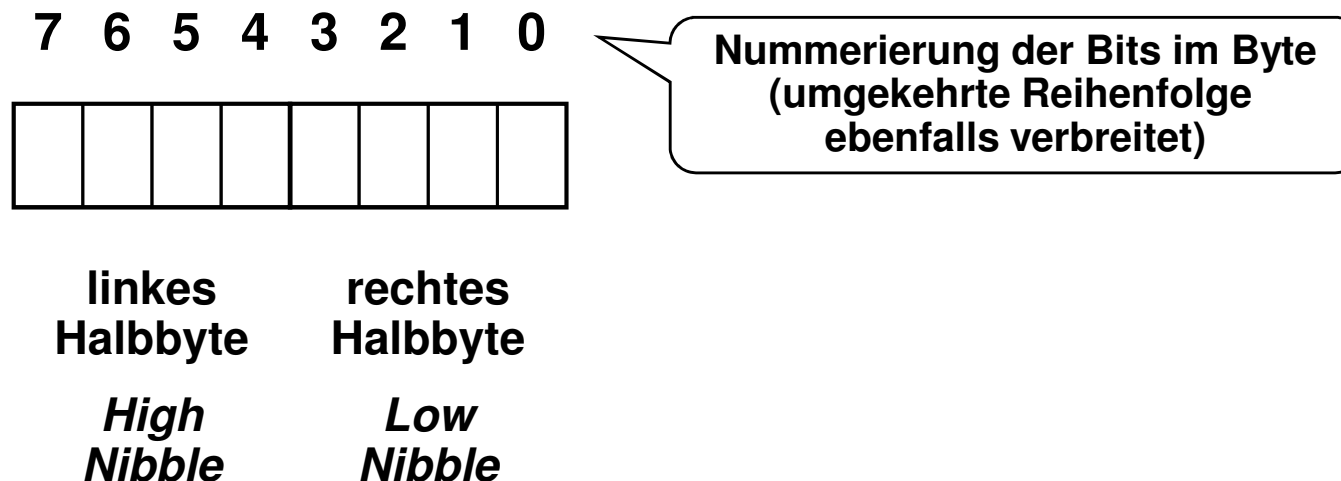
- Wiederholung (vgl. Kap. 2.4): Für $\Sigma = \{0,1\}$ heißen die Elemente der Menge Σ^n n-Bit-Wörter oder Binärwörter der Länge n.
- Informationen werden i.d.R. nicht auf der Bit-Ebene betrachtet, sondern zu größeren Einheiten zusammengefaßt, die bei variabler Länge als *Bitfolgen*, bei fester Länge auch als *Bitvektoren* bezeichnet werden.
- Bezeichnungen für Bitfolgen bestimmter Länge:

<i>Länge n in Bit</i>	<i>Bezeichnung</i>	<i>Anzahl versch. Wörter</i>	<i>Anwendung</i>
1	Bit (b)	2	
3	Triade	$2^3=8$	Oktalziffern (vgl. 3.3)
4	Tetrade, Halb-Byte, Nibble (auch: nybble)	$2^4=16$	Hexadezimal- und Dezimalziffern (vgl. 3.3)
8	Byte (B), Oktett	$2^8=256$	Zeichen (vgl. 3.4)



Byte

- Die kleinste adressierbare Einheit im Arbeitsspeicher heutiger Rechner ist ein **Byte**:



- Die Bytes im Speicher sind fortlaufend nummeriert, die Positionsnummer eines Bytes im Speicher heißt **Adresse**.



Maschinenwörter

- Die Hardware eines Rechensystems verwaltet i.d.R. nur Binärwörter von wenigen festen Längen. Solche Binärwörter heißen *Maschinenwörter*.
- Günstige Längen von Maschinenwörtern sind durch verschiedene Faktoren bestimmt wie (vgl. Kap. 6):
 - Länge der kleinsten im Speicher adressierbaren Einheit
 - Länge der vom/zum Speicher transferierten Einheiten
 - Längen der Verarbeitungseinheiten (elementaren Datentypen) des Prozessors
 - Breite von Datenpfaden und Ein-/Ausgabe-Schnittstellen



Maschinenwörter (2)

- **Übliche Maschinenwortlängen sind Vielfache von Bytes:**
 - typisch: 32 Bit (4 Bytes) ("32-Bit-Prozessor")
 - aber auch z.B. 8 Bit und 16 Bit bei einfachen Mikroprozessoren / Mikrocontrollern
 - 64 Bit bzw. 128 Bit bei derzeitigen Hochleistungsprozessoren bzw. Graphik-Prozessoren
 - entsprechend: Halbwort, Doppelwort, Vierfachwort.



Bezeichnungen für Maßeinheiten

Abk.	Bezeichnung	=	Vergleich
B	Byte		Zeichen
kB	Kilobyte	$2^{10} \text{ B} = 1.024 \text{ B} \approx 10^3 \text{ B}$	Seite \approx 4.000 Zeichen
MB	Megabyte	$1 \text{ MB} = 2^{20} \text{ B} = 1.048.576 \text{ B} \approx 10^6 \text{ B}$	Arbeitsspeicher, heute ca. 256 – 1024 MB
GB	Gigabyte	$1 \text{ GB} = 2^{30} \text{ B} = 1.073.741.824 \text{ B} \approx 10^9 \text{ B}$	Festplatte, ca. 40-250 GB
TB	Terabyte	$1 \text{ TB} = 2^{40} \text{ B} \approx 10^{12} \text{ B}$	Große Datenbestände in Rechenzentren
PB	Petabyte	...	Datenbestände von Supercomputern der nächsten Generation



Bezeichnungen für Maßeinheiten

Dezimal, SI-konform			IEC-System			Vergleich
Abk.	Bezeichn.	Wert	Abk.	Bezeichn.	Wert	
B	Byte	1	B	Byte	1	Zeichen; 1 – 6 B in UTF8-Codierung
kB	Kilobyte	1 kB = 10^3 B = 1000 B	KiB	Kibibyte	1 KiB = 2^{10} B = 1.024 B	Textseite \approx 4.000 Zeichen
MB	Megabyte	1 MB = 10^6 B = 1000^2 B	MiB	Mebibyte	1 MiB = 2^{20} B = 1.048.576 B	Digitalphoto \approx 2 - 12 MB
GB	Gigabyte	1 GB = 10^9 B = 1000^3 B	GiB	Gibibyte	1 GiB = 2^{30} B = 1.073.741.824 B	2,5"-Festplatte, ca. 250 GB – 1 TB
TB	Terabyte	1 TB = 10^{12} B = 1000^4 B	TiB	Tebibyte	1 TiB = 2^{40} B $\approx 1,10 \times 10^{12}$ B	Große Festplatten, Fileserver, RAIDs
PB	Petabyte	1 PB = 10^{15} B = 1000^5 B	PiB	Pebibyte	1 PiB = 2^{50} B $\approx 1,13 \times 10^{15}$ B	Datenbestände von Supercomputern

Logische (Boolesche) Operationen auf Bitvektoren

Basis sind die Booleschen Funktionen

- **ODER** (engl. **OR**) oder **Disjunktion** $ODER : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$,
entspricht in der Mengenlehre der Vereinigungsmenge,
gleichwertige Schreibweisen: $ODER(x,y)$, $OR(x,y)$, $x \vee y$, $x + y$, x / y
- **UND** (engl. **AND**) oder **Konjunktion** $UND : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$,
entspricht in der Mengenlehre der Durchschnittsmenge,
gleichwertige Schreibweisen: $UND(x,y)$, $AND(x,y)$, $x \wedge y$, $x * y$, $x \& y$
- **NICHT** (engl. **NOT**) oder **Negation** $NICHT : \{0,1\} \rightarrow \{0,1\}$,
entspricht in der Mengenlehre der Komplementmenge,
gleichwertige Schreibweisen: $NICHT(x)$, $NOT(x)$, $\neg x$, \bar{x}
- **Übliche Bedeutung:** $0 = falsch (false)$, $1 = wahr (true)$
- Eine Boolesche Funktion wird durch eine sogenannte **Wahrheitstafel** (Wertetabelle) beschrieben, die das Ergebnis der Funktion für alle möglichen Kombinationen der binären Eingabewerte festlegt.



Wahrheitstafeln für ODER, UND, NICHT

ODER (OR): $x \vee y$

$\vee : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

UND (AND): $x \wedge y$

$\wedge : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

NICHT (NOT): $\neg x$

$\neg : \{0,1\} \rightarrow \{0,1\}$

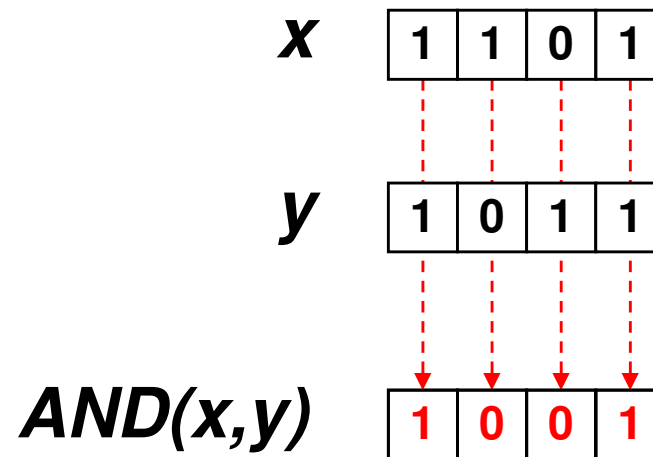
x	$\neg x$
0	1
1	0

0 = falsch (false)

1 = wahr (true)

Durchführung der logischen Operationen

- Gleichzeitiges (paralleles) Anwenden der geforderten Booleschen Funktion auf alle korrespondierenden Bitstellen
- Beispiel:





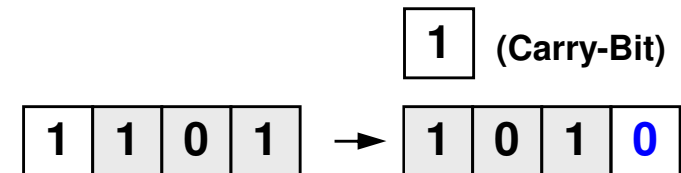
Weitere Operationen auf Bitvektoren

- Unterstützung in vielen Prozessoren für

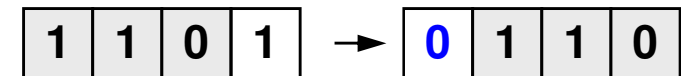
- Setzen und Löschen einzelner Bits

- Test eines Bits, ob gesetzt

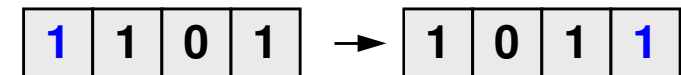
- Verschieben nach links (shift left)



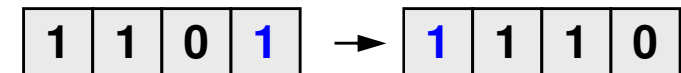
- Verschieben nach rechts (shift right)



- Rotieren nach links (rotate left)



- Rotieren nach rechts (rotate right)



Bit-Operationen in Programmiersprachen

- In Programmiersprachen wie C/C++, Java, Ruby sind Bit-Operationen für ganze Zahlen mit folgenden Operatoren implementiert:

int x = 13; /* Bitmuster: 0...01101₂ = 13₁₀ */

int y = 3; /* Bitmuster: 0...00011₂ = 3₁₀ */

Negation: ~x; /* 1...10010₂ = -14₁₀ */

1x rechts schieben: x >> 1; /* 0...00110₂ = 6₁₀ */

Zweier-Komplement,
behandeln wir in
Kap. 3.3.3

1x links schieben: x << 1; /* 0...11010₂ = 26₁₀ */

AND: x & y; /* 0...00001₂ = 1₁₀ */

OR: x | y; /* 0...01111₂ = 15₁₀ */

XOR: x ^ y; /* 0...01110₂ = 14₁₀ */

3.3 Zahlensysteme, Zahlendarstellungen, Arithmetik

- Betrachtung von zweckmäßigen Darstellungen für Zahlenmengen und zugehörigen arithmetischen Operationen
- Problem: Die aus der Mathematik üblichen Zahlenbereiche (natürliche Zahlen \mathbb{N} , ganze Zahlen \mathbb{Z} , rationale Zahlen \mathbb{Q} , reelle Zahlen \mathbb{R}) sind unendlich, die Anzahl der möglichen Codewörter in Maschinenwörtern fester Länge aber endlich.

⇒ **Exaktes Rechnen nur mit beschränkten Zahlenmengen,**
oder:
Approximative (angenäherte) Zahlendarstellungen und Operationen



Überblick

1. Grundlagen Zahlensysteme
2. Konvertierung von Zahlenwerten
3. Darstellung ganzer Zahlen
4. Darstellung von Festkommazahlen
5. Darstellung von Gleitkommazahlen



3.3.1 Zahlensysteme

- Grundlage:
Stellenwertsysteme (*polyadische Systeme*, *B-adische Systeme*)
- Eine natürliche Zahl $n \in \mathbb{N}$ kann dargestellt werden durch

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

Dabei ist

- B die **Basis** des Zahlensystems $B \in \mathbb{N}$, $B \geq 2$,
- b_i sind Zahlenkoeffizienten $b_i \in \{0, \dots, B-1\}$ (Ziffern),
- nur endlich viele b_i sind $\neq 0$; sei N der größte Index.



3.3.1 Zahlensysteme (2)

- In der Kurzform werden nur die signifikanten Koeffizienten notiert:

$$n = (b_N b_{N-1} b_{N-2} \dots b_1 b_0)_B$$

Angabe der Basis B

d.h. führende Nullen werden unterdrückt.

- **Jede natürliche Zahl lässt sich zu jeder Basis B darstellen.**
- Die Koeffizientenfolge (b_i) ist bei gegebener Basis B eindeutig bestimmt.



Wichtige B-adische Zahlensysteme

- Die für die Informatik wichtigsten Basen

B=10 Dezimalsystem Ziffern 0, ..., 9

B=2 Dual-/Binärsystem Ziffern 0, 1

B=8 Oktalsystem Ziffern 0, ..., 7

B=16 Hexadezimalsystem Ziffern 0, ..., 9, A, B, C, D, E, F
Sedezimalsystem

A 10_{10}
B 11_{10}
C 12_{10}
D 13_{10}
E 14_{10}
F 15_{10}

- Beispiel: Unterschiedliche Darstellungen der gleichen Zahl 28_{10}

$$\begin{aligned} 28_{10} &= 2 \cdot 10^1 + 8 \cdot 10^0 \\ 103_5 &= 1 \cdot 5^2 + 0 \cdot 5^1 + 3 \cdot 5^0 \\ 11100_2 &= 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \\ 34_8 &= 3 \cdot 8^1 + 4 \cdot 8^0 \\ 1C_{16} &= 1 \cdot 16^1 + C \cdot 16^0 \end{aligned}$$

= 12_{10}



Anmerkungen

- **Je größer die Basis ...**
 - um so weniger Ziffern benötigt man zur Darstellung einer Zahl
 - um so schwieriger ist das "kleine Einmaleins" ($1...B$ mal $1...B$)
- **Bedeutung des Hexadezimalsystems**
 - besser lesbar als Dualdarstellung derselben Zahl
 - unmittelbare Umwandlung zum/vom Dualsystem (vgl. auch 3.3.2)
 - breite Anwendung in der Datenverarbeitung
 - Angabe von Arbeitsspeicheradressen
 - Inhalte von Maschinenwörtern
 - Operanden für Bitfeld-Operationen
 - C-Notation: Beispiel 0xFFFF
 - ...



Darstellung von Brüchen

- Ein Bruch kann durch negative Exponenten dargestellt werden:

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

Testfrage:
Wieso sprechen wir von
Darstellung bei \mathbb{Q} , aber „nur“
von *Approximation* bei \mathbb{R} ?

- Zusammengefasst ergibt sich damit zur Darstellung rationaler Zahlen (und zur Approximation reeller Zahlen):

$$x = \sum_{i=-M}^N b_i B^i = \sum_{i=-M}^{-1} b_i B^i + \sum_{i=0}^N b_i B^i$$

- bzw. in der Kurzform der signifikanten Koeffizienten (Ziffern):

$$x = (b_N b_{N-1} b_{N-2} \dots b_1 b_0 , b_{-1} b_{-2} \dots b_{-M})_B$$

Komma



Horner-Schema

- andere Schreibweise, bedeutend wegen vereinfachter Berechnungsweise (vgl. 3.3.2)
- für natürliche Zahlen:

$$n = \sum_{i=0}^N b_i B^i = ((\dots (b_N * B + b_{N-1}) * B + \dots + b_2) * B + b_1) * B + b_0$$

Bsp.: $4312 = 4*10^3 + 3*10^2 + 1*10^1 + 2*10^0 = ((4*10 + 3) * 10 + 1) * 10 + 2$

- für den gebrochenen Anteil:

$$z = \sum_{i=-M}^{-1} b_i B^i = ((\dots (b_{-M} / B + b_{-M+1}) / B + \dots + b_{-2}) / B + b_{-1}) / B$$

Bsp.: $0,342 = 2*10^{-3} + 4*10^{-2} + 3*10^{-1} = ((2 / 10 + 4) / 10) + 3) / 10$



3.3.2 Konvertierung von Zahlenwerten

- **Aufgabe**
 - (a) Gegeben sei eine natürliche Zahl $n \in \mathbb{N}$ zur Ausgangsbasis B' (hier häufig 10 oder 2).
Bestimme die Zahlendarstellung von n zur Zielbasis B (hier häufig 2 oder 10).
 - (b) analog für Brüche
- **Methoden basieren auf:**
 - Potenzreihendarstellung
 - Horner-Schema
- **Unterscheidung**
 - Rechnen im Ausgangssystem
 - Rechnen im Zielsystem

Division durch fallende Potenzen der Zielbasis

- Rechnen im Ausgangssystem
- Grundlage:

$$n = \sum_{i=0}^N b_i B^i = b_N * B^N + b_{N-1} * B^{N-1} + \dots + b_2 * B^2 + b_1 * B + b_0$$

- Verfahren
 - (1) Bestimme höchste Potenz N mit $B^N \leq n$
 - (2) Bestimme Koeff. $b_N = \lfloor n / B^N \rfloor$ ($\lfloor x \rfloor :=$ größte ganze Zahl $\leq x$)
 - (3) Bilde Rest $R_{N-1} := n - b_N * B^N$
 - (4) Betrachte analog die nächst kleinere Potenz i (N-1) und bestimme den Koeff. $b_i = \lfloor R_i / B^i \rfloor$
 - (5) Setze $R_{i-1} := R_i - b_i * B^i$
 - (6) Wiederhole (4) und (5) bis $i=0$
 - (7) $(b_N b_{N-1} \dots b_2 b_1 b_0)_B$ ist die Darstellung von n zur Basis B.

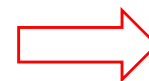


Beispiel 1

- Konvertiere 122_{10} zur Basis $B=2$
- $N = 6$

R_i	$/ B^i$	b_i
$n=122$	$2^6=64$	$b_6= 1$
$122-64=58$	$2^5=32$	$b_5= 1$
$58-32=26$	$2^4=16$	$b_4= 1$
$26-16=10$	$2^3=8$	$b_3= 1$
$10-8=2$	$2^2=4$	$b_2= 0$
$2-0=2$	$2^1=2$	$b_1= 1$
$2-2=0$	$2^0=1$	$b_0= 0$

$$\begin{aligned} 2^8 &= 256 \\ 2^7 &= 128 \\ 2^6 &= 64 \\ 2^5 &= 32 \\ 2^4 &= 16 \\ 2^3 &= 8 \\ 2^2 &= 4 \\ 2^1 &= 2 \\ 2^0 &= 1 \end{aligned}$$



$$122_{10} = 1111010_2$$



Beispiel 2

- Konvertiere $n=122_{10}$ zur Basis $B=5$
- $N =$

R_i	$/ B^i$	b_i
$n=122$		

$$5^4 = 625$$

$$5^3 = 125$$

$$5^2 = 25$$

$$5^1 = 5$$

$$5^0 = 1$$



Beispiel 2

- Konvertiere $n=122_{10}$ zur Basis $B=5$
- $N = 2$

R_i	$/ B^i$	b_i
$n=122$	$5^2=25$	$b_2= 4$
$122-4*25=22$	$5^1=5$	$b_1= 4$
$22-4*5=2$	$5^0=1$	$b_0= 2$

$$\Rightarrow 122_{10} = 442_5$$

$$\begin{aligned} 5^4 &= 625 \\ 5^3 &= 125 \\ 5^2 &= 25 \\ 5^1 &= 5 \\ 5^0 &= 1 \end{aligned}$$

✱ Horner-Schema für natürliche Zahlen (a)


- Grundlage:

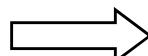
$$n = \sum_{i=0}^N b_i B^i = ((\dots (b_N * B + b_{N-1}) * B + \dots + b_2) * B + b_1) * B + b_0$$

(a) Rechnen im Ausgangssystem

- Verfahren: schrittweise Division von $n \in \mathbb{N}$ durch die Zielbasis B

Schritt	Division	Quotient	Rest
1	n / B	$(\dots(b_N * B + b_{N-1}) * B + \dots + b_2) * B + b_1$	b_0
2	$(n/B) / B$	$\dots(b_N * B + b_{N-1}) * B + \dots + b_2$	b_1
...
N	n / B^N	b_N	b_{N-1}
N+1	n / B^{N+1}	0	b_N



 $n = (b_N \ b_{N-1} \ \dots \ b_2 \ b_1 \ b_0)_B$



Beispiel 1

- Konvertiere 122_{10} zur Basis $B=2$
- Rechnen im Quellsystem (hier Dezimalsystem)

Schritt	/ B	Quotient	Rest
122	/2	61	0
61	/2	30	1
30	/2	15	0
15	/2	7	1
7	/2	3	1
3	/2	1	1
1	/2	0	1



Ablesefolge der Ziffern

$$\Rightarrow 122_{10} = 1111010_2$$



Beispiel 2

- Konvertiere 122_{10} zur Basis $B=5$
- Rechnen im Quellsystem (hier Dezimalsystem)

Schritt	/ B	Quotient	Rest
122	/5	24	2
24	/5	4	4
4	/5	0	4

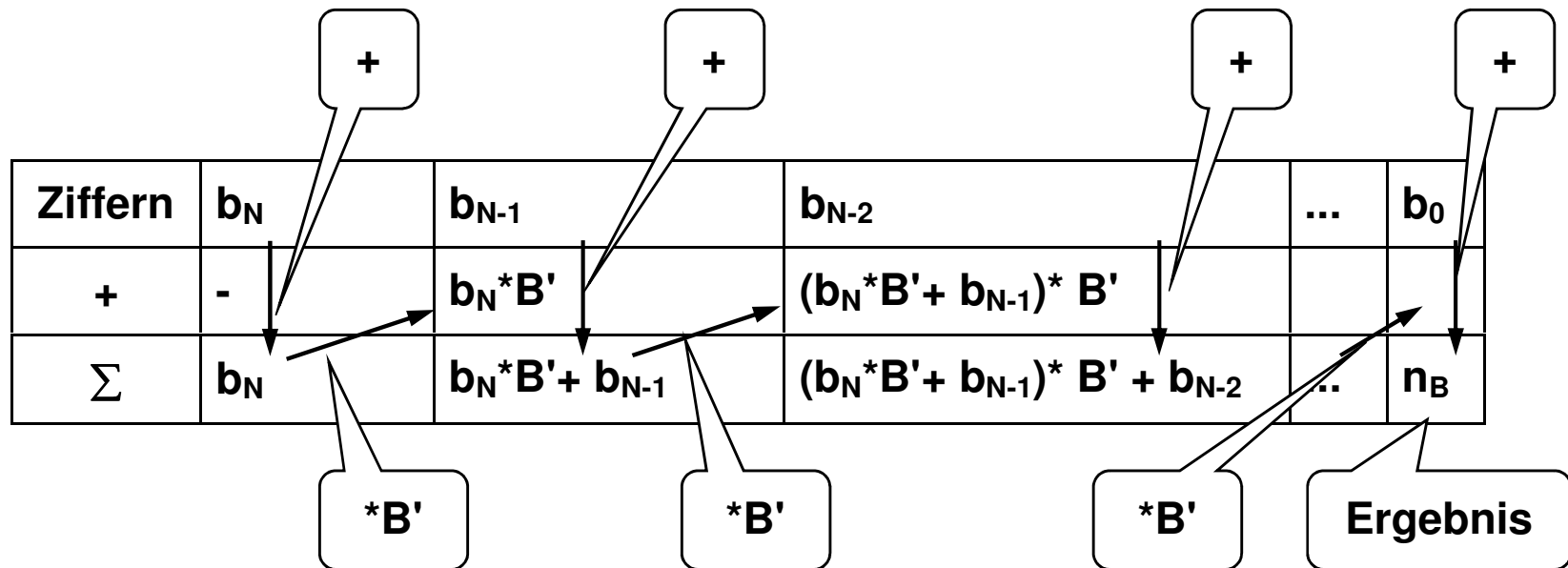


$$\Rightarrow 122_{10} = 442_5$$

✱ Horner-Schema für natürliche Zahlen (b)

(b) Rechnen im Zielsystem

- Verfahren:
 - Darstellen aller Ziffern und der Ausgangsbasis B' im Zielsystem
 - Auswerten des Horner-Schemas von "innen" nach "außen"

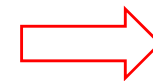




Beispiel 1

- Konvertiere 1111010_2 zur Basis $B=10$
- Rechnen im Zielsystem (hier Dezimalsystem)
- $B'=2_{10}$

Ziffern	1	1	1	1	0	1	0
+	–	$1*2$	$3*2$	$7*2$	$15*2$	$30*2$	$61*2$
Σ	1	3	7	15	30	61	122



$$1111010_2 = 122_{10}$$



Beispiel 2

- Konvertiere $2AC_{16}$ zur Basis $B=10$
- Rechnen im Zielsystem (hier Dezimalsystem)
- $B'=16_{10}$

		A_{16}	C_{16}
Ziffern	2	10	12
+	–	$2 \cdot 16$	$42 \cdot 16$
Σ	2	42	684

$$\Rightarrow 2AC_{16} = 684_{10}$$



Konvertierung gebrochener Zahlen

- **Getrennte Behandlung von Vorkommateil und Nachkommateil**
- **Vorkommateil ist natürliche Zahl, Umwandlung wie beschrieben**
- **Umwandlung des Nachkommateils entsprechend dem Horner-Schema für Brüche (nächste Folie)**
- **Weitere Probleme**
 - Anzahl der notwendigen Schritte nicht von vornherein bekannt
 - Ein gegebener endlicher Bruch muss zur Zielbasis nicht endlich sein.
 - Beispiel: $1/3 = 3^{-1} = 0,1_3 = 0,333333..._{10}$



Horner-Schema für Brüche (a)

- Grundlage:

$$z = \sum_{i=-M}^{-1} b_i B^i = ((\dots (b_{-M} / B + b_{-M+1}) / B + \dots + b_{-2}) / B + b_{-1}) / B$$

(a) Rechnen im Ausgangssystem

- Verfahren: schrittweise Multiplikation von $z \in \mathbb{Q}$ mit der Zielbasis B

Schritt	Mult.	Produkt (Bruchteil)	Ganzteil b_i
1	$z * B$	$(\dots(b_{-M}/B + b_{-M+1})/B + \dots + b_{-3})/B + b_{-2})/B$	b_{-1}
2	$(z*B)*B$	$(\dots(b_{-M}/B + b_{-M+1})/B + \dots + b_{-3})/B$	b_{-2}
...
M-1	$z * B^{M-1}$	b_{-M}/B	b_{-M+1}
M	$z * B^M$	0	b_{-M}

$$\Rightarrow z = (0, b_{-1} b_{-2} \dots b_{-M})_B$$



Beispiel

- Konvertiere $0,21_{10}$ zur Basis $B=2$
- Rechnen im Quellsystem (hier Dezimalsystem)

Wert	* B	Produkt (Zwischenerg.)	Bruchteil	Ganzteil
0,21	*2	0,42	0,42	0
0,42	*2	0,84	0,84	0
0,84	*2	1,68	0,68	1
0,68	*2	1,36	0,36	1
0,36	*2	0,72	0,72	0
0,72	*2	1,44	0,44	1
0,44	*2	0,88

$$\Rightarrow 0,21_{10} = 0,001101..._2$$



Horner-Schema für Brüche (b)

(b) Rechnen im Zielsystem

- entspricht dem Vorgehen bei natürlichen Zahlen mit Division statt Multiplikation
- Verfahren:
 - Darstellen aller Ziffern und der Ausgangsbasis B' im Zielsystem
 - Auswerten des Horner-Schemas von "innen" nach "außen":
 - (1) kleinstwertige Ziffer durch die Ausgangsbasis in deren Zieldarstellung dividieren
 - (2) Ergebnis zur nächsthöherwertigen Stelle addieren
 - (3) Ergebnis aus (2) wieder durch Ausgangsbasis in deren Zieldarstellung dividieren
 - (4) wiederholen (2) und (3), bis alle Stellen verarbeitet sind



Beispiel

- Konvertiere $0,01011_2$ zur Basis $B=10$
- Rechnen im Zielsystem (hier Dezimalsystem)
- $B'=2_{10}$

Ziffern	1	1	0	1	0	,
+	-	$1/2=0,5$	$1,5/2=0,75$	$0,75/2=0,375$	$1,375/2=0,6875$	$0,6875/2=0,34375$
Σ	1	1,5	0,75	1,375	0,6875	0,34375

$\Rightarrow 0,01011_2 = 0,34375_{10}$

Schnelle Konvertierung bei "verwandten" Basen

- Für die Praxis von Bedeutung ist die schnelle Konvertierung zwischen Dual-, Oktal-, und Hexadezimalsystem
- Gilt allgemeiner für Basen B' und B mit $B = B'^k$, k ganzzahlig.
- Dann lassen sich Gruppen von k Ziffern der Darstellung zur Basis B' zusammenfassen zu einer Ziffer der Basis B , beginnend am Komma nach links und rechts.
- Beispiele:

$$1111010_2 = 001 | 111 | 010_2 = 172_8$$

$$1111010_2 = 111 | 1010_2 = 7A_{16}$$

$$2BC_{16} = 0010 | 1011 | 1100_2 = 001 | 010 | 111 | 100_2 = 1274_8$$

$$1101001,11011_2 = 001 | 101 | 001,110 | 110_2 = 151,66_8$$

$$1101001,11011_2 = 0110 | 1001,1101 | 1000_2 = 69,D8_{16}$$



Arithmetik in Stellenwertsystemen

- In Stellenwertsystemen beliebiger Basis B lassen sich prinzipiell die Verfahren des schriftlichen Rechnens anwenden, wie man sie vom Dezimalsystem her kennt
 - Addition mit Übertrag
 - Subtraktion mit Borgen
 - Multiplikation mit stellenrichtiger Addition
 - Division mit Rest

- **Beispiel:**

$$\begin{array}{r} 24, 13_5 \\ + 12, 34_5 \\ \hline 42, 02_5 \end{array}$$

$$\begin{array}{r} 24, 13_5 \\ - 12, 34_5 \\ \hline 11, 24_5 \end{array}$$

$$\begin{array}{r} 24 * 13_5 \\ \hline 24 \\ 132 \\ \hline 422_5 \end{array}$$

$$\begin{array}{r} 10223_5 : 13_5 = 321_5 \\ 44 \\ \hline 32 \\ 31 \\ \hline 13 \\ 13 \end{array}$$

- Die in Rechensystemen angewendete Arithmetik wird im folgenden Abschnitt behandelt.



3.3.3 Darstellung ganzer Zahlen

- Rechensysteme verwenden das Dual-/Binärsystem zur Speicherung und Verarbeitung aller Zahlendarstellungen.
- Maschinenwörter werden zur Aufnahme von Zahlendarstellungen verwendet.
- Die Wortlänge (Wortbreite) legt die Anzahl der darstellbaren Zahlenwerte fest:

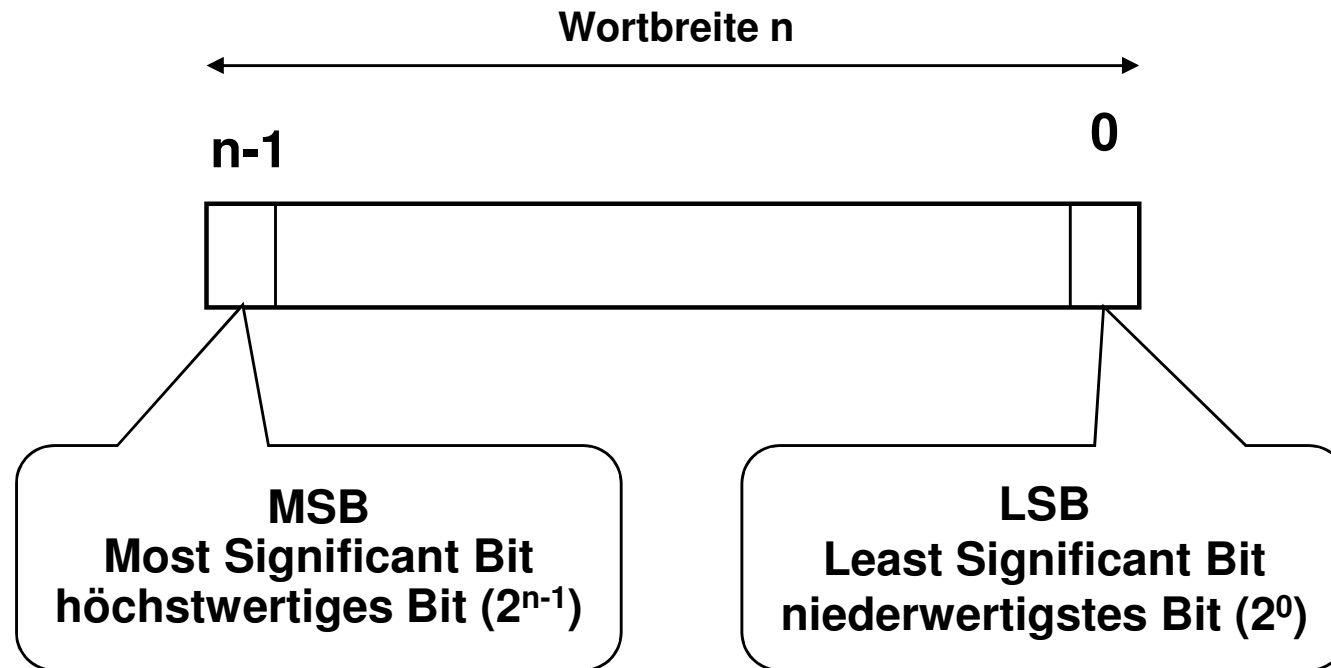
Wortlänge [bit]	Anzahl Darstellungen
8	$2^8 = 256$
16	$2^{16} = 65536$
32	$2^{32} \approx 4.3 \cdot 10^9$
64	$2^{64} \approx 1.8 \cdot 10^{19}$

- Vorrangiges Ziel: Exaktes Rechnen mit ganzen Zahlen
⇒ Umgang nur mit beschränkten Zahlenmengen möglich
-



Bit-Positionen in Maschinenwörtern

- Festlegung ausgezeichneter Bit-Positionen in n-Bit-Maschinenwörtern (typisch):





Byte Ordering

- Technisch sind zwei unterschiedliche Adressierungsweisen der Bytes in einem Maschinenwort möglich:
 - **Big-Endian:** höherwertige Stelle in Byte mit niedriger Adresse.
Beispiele: Sun SPARC, Motorola, IBM Mainframe
 - **Little-Endian:** niederwertige Stelle in Byte mit niedriger Adresse.
Beispiele: Intel x86, DEC VAX, ARM (Standard)

Byte-Adressen

n+3

n+2

n+1

n

Big-Endian

224	36	175	193
-----	----	-----	-----

höchstwertiges Byte

Little-Endian

193	175	36	224
-----	-----	----	-----

höchstwertiges Byte



Vorzeichenlose ganze Zahlen

- Verwendung
 - für einen Bereich natürlicher Zahlen \mathbb{N} einschl. 0
 - sowie für Adressen von Speicherwörtern
- Programmiersprachenebene: *unsigned integer*
- Nutzung des gesamten Maschinenworts zur Darstellung der Zahl in Dualdarstellung

Wortlänge [bit]	Wertebereich	C (typisch)
n	$0 \dots 2^n - 1$	
8	0 ... 255	(unsigned char)
16	0 ... 65535	unsigned short int
32	0 ... 4.294.967.295	unsigned int
64	$0 \dots 2^{64} - 1$	unsigned long int



Vorzeichenbehaftete ganze Zahlen

Überblick:

- Verwendung für einen Bereich ganzer Zahlen \mathbb{Z}
- Programmiersprachenebene: *signed integer*
- Alternativen zur Darstellung
 1. Vorzeichen/Betrags-Darstellung
 2. Excess-Darstellung
 3. Komplementdarstellung
 - B-1 - Komplement
 - B - Komplement
- Die verschiedenen Alternativen besitzen jeweils Vor- und Nachteile.
- In Hinblick auf die technische Realisierung der Arithmetik in einem Prozessor (Arithmetisch-Logische Einheit, vgl. Kap. 6) besitzt die Komplementdarstellung die meisten Vorteile.

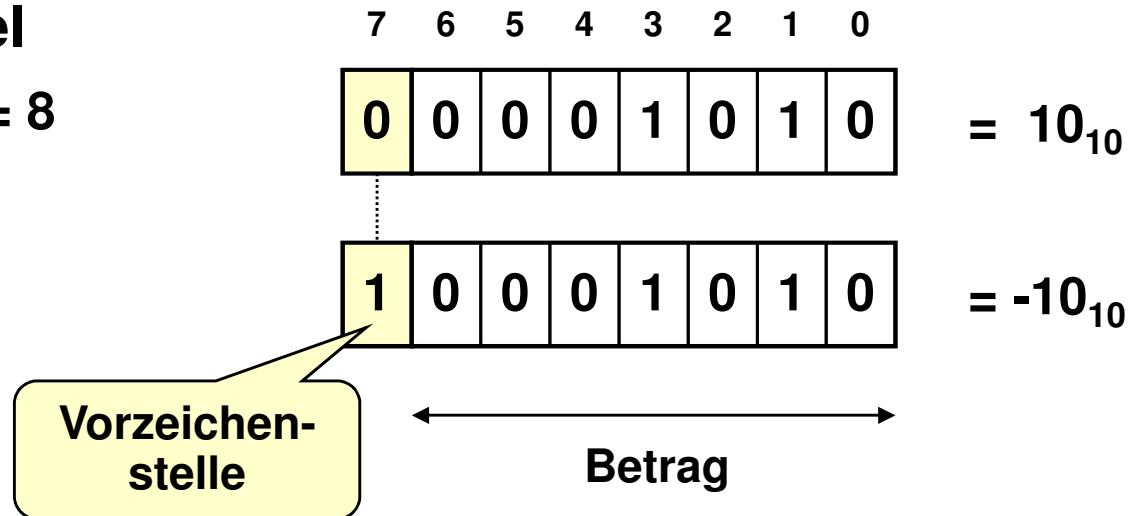


Vorzeichen/Betrags-Darstellung

- Zahlendarstellung mit Vorzeichen-Bit (engl. *sign bit*) und Betrag in einem n-Bit-Maschinenwort
 - Nutzung des höchstwertigen Bits (MSB) zur Aufnahme des Vorzeichens von z
 - MSB = 0: $z \geq 0$
 - MSB = 1: $z < 0$
 - Nutzung der restlichen Bits des Wortes zur Dualdarstellung des Betrags von z.

- **Beispiel**

- $n = 8$





Vorzeichen/Betrags-Darstellung (2)

- **Konsequenzen**
 - Zahlenbereich $-(2^{n-1} - 1) \dots (2^{n-1} - 1)$
 - Zwei Darstellungen der "Null" (00...0 und 10...0, genannt **0 und -0**)
 - Unterschiedliche Behandlung von Subtraktion und Addition, Fallunterscheidungen
 - relativ hoher Hardware-Aufwand wäre notwendig

⇒ in realen Prozessoren nicht verwendet



Excess-Darstellung

Def

- Sei z eine ganze Zahl. Dann heißt $z' = z + k$ die **Excess- k -Darstellung** von z (Hinzuhaddieren eines festen Betrags (*Excess*)).
- Anwendung:
 - Maschinenwortlänge n
 - $-2^{n-1} \leq z \leq 2^{n-1}-1$, $k = 2^{n-1}$
 - $\Rightarrow 0 \leq z' \leq 2^n-1$
- Beispiel
 - $n = 8$, $k = 2^{n-1} = 128$
 - $-128 \leq z \leq 127$
 - $z' = z+128$ ist die Excess-128-Darstellung von z
 - $\Rightarrow 0 \leq z' \leq 255$



Excess-Darstellung (2)

- **Vorteile/Nachteile**
 - Inkrementieren/Dekrementieren wie bei vorzeichenlosen ganzen Zahlen
 - Ordnungsbeziehung bleibt erhalten:
$$y' \leq z' \Rightarrow y \leq z$$
 - Korrektur bei Addition/Subtraktion notwendig („mod $2k$ “):
$$y' + z' = (y + k) + (z + k) = ((y + z) + k) + k = (y + z)' + k \neq (y + z)'$$
- **Anwendung**
 - Exponentendarstellung von Gleitpunktzahlen (vgl. 3.3.5)
 - Analog/Digital- und Digital/Analog-Wandler (vgl. 3.5)



Komplemente

Vorbemerkungen

- Bei endlicher Wortlänge ist das Komplement einer Zahl vergleichbar mit dem additiv inversen Element.
- Da gilt: Subtraktion = Addition mit dem additiv inversen Element, genügt Addition und Komplementbildung zur Subtraktion!
- Stellenkomplemente sind besonders einfach zu bilden. Normale Komplementbildung wird daher auf die Bildung von Stellenkomplementen zurückgeführt.
- Wichtig für Hardware-Entwickler:
 - Addition (mühsam) und Stellenkomplement (einfach) einzubauen genügt, um auch subtrahieren zu können!



Komplement-Bildung

Def

- Sei B Basis eines Stellenwertsystems, n die betrachtete Wortlänge, z eine ganze Zahl zur Basis B.
Das **B-Komplement** $^{(B)}z$ von z wird definiert durch $z + ^{(B)}z = B^n$.

- Es gilt:

$$^{(B)}z = B^n - z = 1 + \underbrace{\sum_{i=0}^{n-1} (B-1) B^i}_{B^n} - \underbrace{\sum_{i=0}^{n-1} z_i B^i}_z = 1 + \underbrace{\sum_{i=0}^{n-1} ((B-1) - z_i) B^i}_{^{(B-1)}z} = 1 + ^{(B-1)}z$$

$^{(B-1)}z$ heißt **(B-1)-Komplement** oder **Stellenkomplement**

- Beispiel (B=10, n=2, z=85):

$$^{(10)}85 = 10^2 - 85 = 1 + (9 \cdot 10 + 9) - 85 = 1 + [(9-8) \cdot 10 + (9-5) \cdot 1] = 1 + 14 = 15$$

- Es gilt damit: $z + ^{(B-1)}z = B^n - 1$ sowie $^{(B)}z = ^{(B-1)}z + 1$

- Das (B-1)-Komplement ergibt sich also durch stellenweise Komplement-Bildung zur größten Ziffer (B-1) der betrachteten Basis.
- Das B-Komplement ergibt sich aus dem (B-1)-Komplement durch Addition von 1.



Beispiel 1

- **Betrachtung der Basis $B=10$ (Dezimalsystem)**
 - **($B-1$)-Komplement:** Neuner-Komplement (Ergänzung zu 9)
 - **B -Komplement:** Zehner-Komplement

- **Beispiel**

$n=6$ Stellen

$$z = 003910_{10}$$

$$^{(9)}z = 996089_{10}$$

+1

$$^{(10)}z = 996090_{10}$$

stellenweise Ergänzung zu $B-1=9$

$$^{(10)}z = ^{(9)}z + 1$$

$$z + ^{(10)}z = 10^6$$



Beispiel 2

- Betrachtung der Basis $B=2$ (Dualsystem)
 - $(B-1)$ -Komplement: Einer-Komplement (Ergänzung zu 1)
 - B -Komplement: Zweier-Komplement
- Einer-Komplement entspricht stellenweiser Invertierung
(in Hardware sehr einfach zu implementieren!)
- Beispiel

$n=8$ Stellen

$$z = 00111010_2$$

$$^{(1)}z = 11000101_2$$

$$+ 1_2$$

$$^{(2)}z = 11000110_2$$

stellenweise Invertierung

$$^{(2)}z = ^{(1)}z + 1$$



Einer-Komplement-Darstellung ganzer Zahlen

- **Zahlendarstellung in einem n-Bit-Maschinenwort**
- **Aufteilung des Darstellungsbereichs in 2 Hälften:**
 - die nicht-negativen ganzen Zahlen $0, \dots, 2^{n-1}-1$ werden in Dualdarstellung wie in der Vorzeichen/Betragsweise dargestellt.
 - die negativen ganzen Zahlen $-(2^{n-1}-1), \dots, -0$ werden durch das Einer-Komplement der betragsgleichen positiven Zahl dargestellt.
- **Vorteil:**
 - symmetrischer Bereich dargestellter Zahlen
- **Nachteile:**
 - doppelte Darstellung der Null (00...0 und 11...1)
 - Korrektur bei Addition/Subtraktion erforderlich, z.B.
$${}^{(1)}y + {}^{(1)}z = B^n - y - 1 + B^n - z - 1 = B^n + B^n - ((y+z) - 1) - 1 = {}^{(1)}(y+z) - 1 \neq {}^{(1)}(y+z)$$

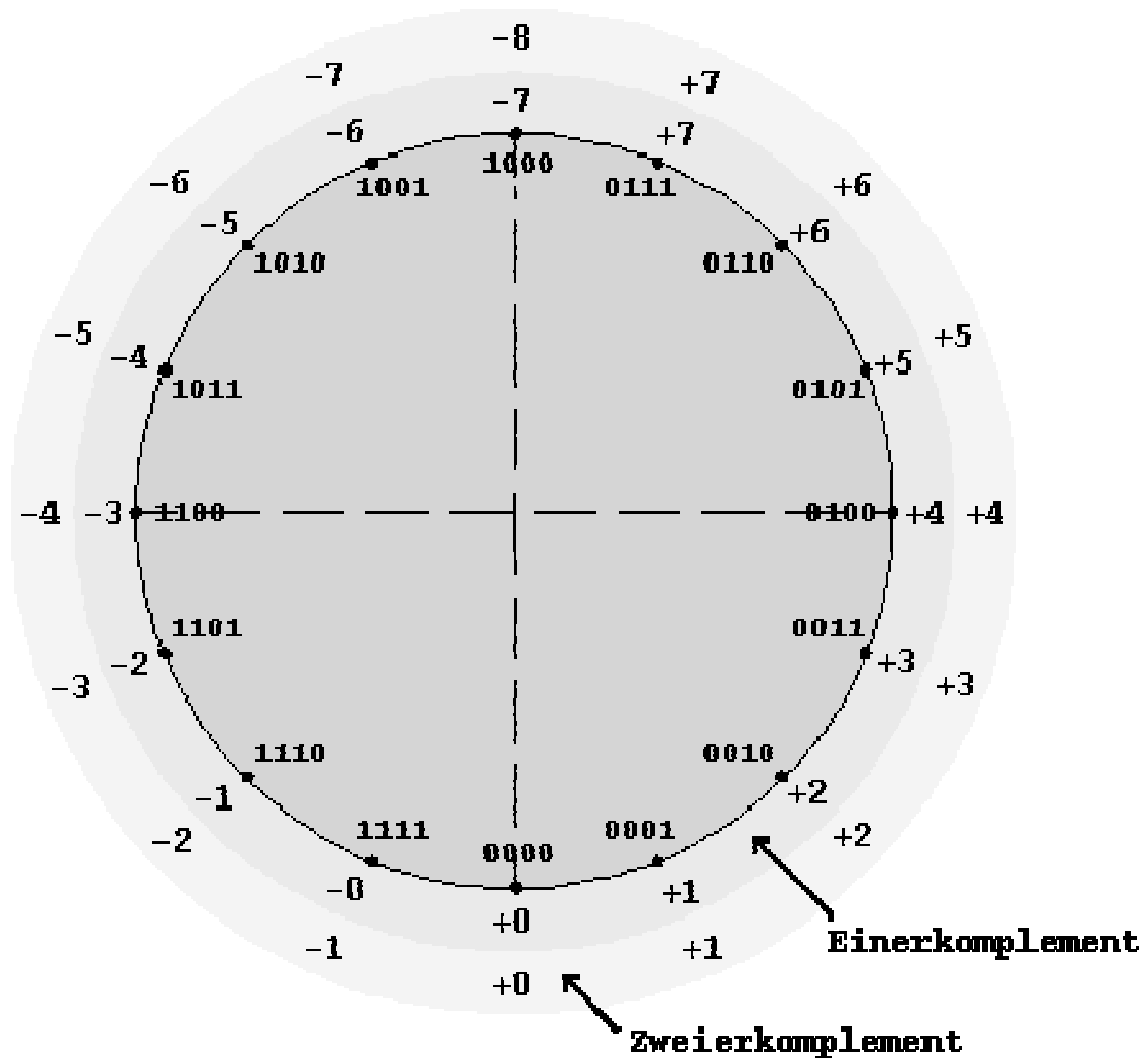
Zweier-Komplement-Darstellung ganzer Zahlen

- **Zahlendarstellung in einem n-Bit-Maschinenwort**
- **Aufteilung des Darstellungsbereichs in 2 Hälften:**
 - die nicht-negativen ganzen Zahlen $0, \dots, 2^{n-1}-1$ werden in Dualdarstellung wie bisher dargestellt.
 - die negativen ganzen Zahlen $-2^{n-1}, \dots, -1$ werden durch das Zweier-Komplement der betragsgleichen positiven Zahl dargestellt und belegen dadurch den Bereich $2^{n-1}, \dots, 2^n-1$.
- **Nachteil:**
 - asymmetrischer Bereich dargestellter Zahlen (Absolutbetrag von -2^{n-1} nicht darstellbar)
- **Vorteile:**
 - Beseitigung der Nachteile der Einer-Komplement-Darstellung
 - Vorzeichen einer Zahl ist weiter am MSB ablesbar:
Eine Zahl ist negativ $\Leftrightarrow \text{MSB}=1$
 - Einfache Arithmetik (s.u.) !!



Verdeutlichung

- Zahlenkreis (n=4)





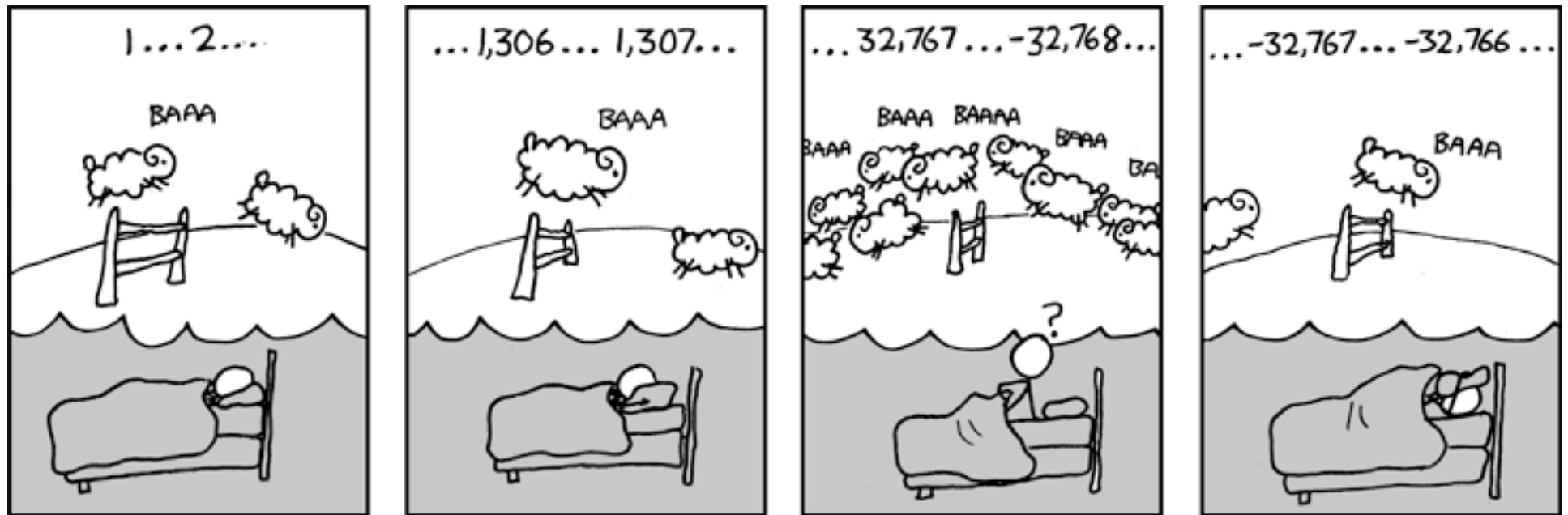
Wertebereiche für ganze Zahlen

- **Betrachtung**
 - n-Bit Maschinenwörter
 - Zweier-Komplement-Darstellung ganzer Zahlen
- **Programmiersprachenebene: *signed integer***

Wortlänge [bit]	Wertebereich	C (typisch)
n	$-2^{n-1} \dots 2^{n-1}-1$	
8	-128 ... 0 ... 127	(signed char)
16	-32768 ... 0 ... 32767	signed short int
32	-2.147.483.648 ... 2.147.483.647	signed int
64	$-2^{63} \dots 2^{63}-1$	signed long int

* <https://xkcd.com/571/> zum Zweierkomplement

- „Can't sleep“





Arithmetik ganzer Zahlen

- **Beschränkung auf Zweier-Komplement-Darstellung**
- **Für Addition/Subtraktion wird nur ein Addierwerk benötigt:**
 - Summe $y+z$ zweier ganzer Zahlen kann durch gewöhnliche Addition im Dualsystem gebildet werden.
 - Subtraktion $y-z$ durch Addition des Zweier-Komplements von z :
 - Einerkomplement von z bilden, inkrementieren, nun zu y addieren!
- **Das Vorzeichen-Bit wird wie eine normale Stelle behandelt!**
 - Ein evtl. Übertrag in die $n+1$. Stelle wird nicht weiter beachtet (außer zur Fehlererkennung, siehe nächste Folie).
- **Beispiele ($n=4$):**

$$\begin{array}{rcl} 0011 & = & 3_{10} \\ + 0010 & = & 2_{10} \\ \hline 0101 & = & 5_{10} \end{array}$$

$$\begin{array}{rcl} 0111 & = & 7_{10} \\ + 1010 & = & -6_{10} \\ \hline (1) 0001 & = & 1_{10} \end{array}$$

$$\begin{array}{rcl} 1101 & = & -3_{10} \\ + 1110 & = & -2_{10} \\ \hline (1) 1011 & = & -5_{10} \end{array}$$



Erkennung von Überlauf

- **Überlauf (Overflow):** Das Ergebnis einer Operation ist außerhalb des darstellbaren Zahlenbereichs und damit ungültig.

- Erkennungsregel:

Bei der Addition zweier Zahlen in Zweier-Komplement-Darstellung findet genau dann ein Überlauf statt, wenn die **Überträge (Carry)** in die n. Stelle und in die gedachte (n+1). Stelle **verschieden** sind.

— Bem.: Realisierung durch einfache Hardware-Schaltung möglich!

- Beispiel:

	0110 = 6 ₁₀	1010 = -6 ₁₀	1101 = -3 ₁₀
	+ 0011 = 3 ₁₀	+ 1101 = -3 ₁₀	+ 1110 = -2 ₁₀
	----	----	----
	01	10	11
	1001 = -7 ₁₀	0111 = 7 ₁₀	1011 = -5 ₁₀

Überträge carry bits

ungleiche Überträge = Fehler !

ungleiche Überträge = Fehler !

kein Fehler !



Arithmetik ganzer Zahlen (2)

- **Multiplikation und Division könnten prinzipiell durch fortgesetzte Addition/Subtraktion und Verschieben basierend auf den Betragswerten und evtl. Komplementierung erfolgen.**
- **Rechenbeispiel:**

```
10110*1001
-----
 10110
 10110
-----
11000110
```

```
11011100:10100 = 1011
10100
-----
 11110
 10100
-----
 10100
 10100
-----
    0
```

- **Tatsächlich existieren in den meisten heutigen Prozessoren spezielle zusätzliche Multiplizierwerke, die eine schnelle Multiplikation erlauben.**

3.3.4 Darstellung von Festkommazahlen

- **Prinzip:** In der Zahlendarstellung wird an einer beliebigen aber festen Stelle ein Komma angenommen.
- Zahlendarstellung und Arithmetik könnten weiter binär sein (vgl. 3.3.2).
- **Beispiel:** $5,25_{10} = 2^2 + 2^0 + 2^{-2} = 101,01_2$
- Da das Komma nur gedacht ist, bliebe die Durchführung der Operationen bis auf die extern notwendige Verwaltung des Kommas unverändert.



Probleme

- **Ungenauigkeit bei der Konvertierung**

- Dezimalzahlen mit endlich vielen Nachkommastellen haben häufig keine endliche Dualdarstellung (vgl. 3.3.2)

- **Rundungsfehler**

- Sorgfältige Wahl der Ausführungsreihenfolge und Genauigkeit von Zwischenergebnissen ist notwendig
- Beispiel:
 - 2 Vorkommastellen, 1 Nachkommastelle
 - $(00,2 * 00,3) * 20,0 = 00,0 * 20,0 = 00,0$
 - $00,2 * (00,3 * 20,0) = 00,2 * 06,0 = 01,2 !!!$



Probleme

- Ungenauigkeiten bei kaufmännischen Anwendungen werden oft als nicht akzeptabel angesehen (Buchführung "auf den Cent genau").
- ⇒ Einführung einer Dezimalarithmetik im Dualsystem durch sogenannte **BCD-Zahlen** (*Binary Coded Decimal*).

Festkomma-Prinzip durch gedachte Anzahl von Nachkommastellen



BCD-Darstellung

- Darstellung der Dezimalziffern in 4-stelligen Dualzahlen

Dezimal- ziffer	BCD- Darstellung
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
	1010
	1011
	1100
	1101
	1110
	1111

} ungültig:
Pseudotetraden

- Dezimalzahlen werden ziffernweise in BCD-Darstellung überführt
-



BCD-Arithmetik

- BCD-Zahlen werden wie gewöhnliche Dualzahlen addiert unter Beachtung von zwei Korrekturen:
 - (a) Bei Übertrag in die nächste Tetrade muss Korrekturwert 6 (0110_2) zur ausgehenden Tetrade addiert werden (Differenz der Basen 16 und 10).
 - (b) Tritt Pseudotetrade als Ergebnis auf, so muss Korrekturwert 6 (0110_2) addiert werden.

⇒ Korrekte BCD-Ziffer wird erzeugt

Übertrag in nächsthöhere Stelle wird generiert
(Korrektur nach (a) entfällt)

- **Beispiele:**

$$\begin{array}{rcl} & 1000 & = 8_{10} \\ \text{zu (a)} & + 1001 & = 9_{10} \\ & \text{----} & \\ & 0001 & \\ & + 0110 & = 6_{10} \\ & \text{-----} & \\ & 0001 & 0111 = 17_{10} \end{array}$$

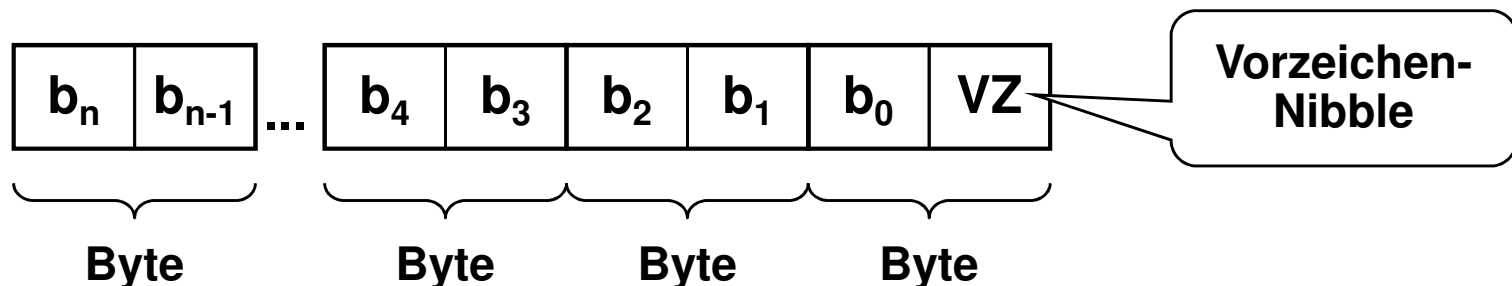
$$\begin{array}{rcl} & 1000 & = 8_{10} \\ \text{zu (b)} & + 0100 & = 4_{10} \\ & \text{----} & \\ & 1100 & = 12_{10} \\ & + 0110 & = 6_{10} \\ & \text{-----} & \\ & 0001 & 0010 = 12_{10} \end{array}$$

Pseudotetrade
Korrekturwert



Beispiel

- Speicherung und Verarbeitung von BCD-Zahlen ist unterschiedlich in verschiedenen Prozessor-Architekturen
- Beispiel (IBM /360, Mainframes):
 - Format: packed decimal
 - je zwei BCD-Dezimalziffern b_i werden in einem Byte (High und Low Nibble) gespeichert.
 - variabel lange Darstellung von 1...31 Dezimalstellen in 1-16 Bytes
 - gedachtes, extern verwaltetes Komma
 - Vorzeichen wird in separatem Nibble dargestellt, gültige Vorzeichen sind alle Pseudotetraden + : CAFE, - : BD
 - separates BCD-Rechenwerk



3.3.5 Darstellung von Gleitkommazahlen

- **Motivation**
 - Numerische Verfahren zur Lösung wiss.-techn. Probleme verlangen häufig einen sehr großen Bereich darstellbarer reeller Zahlen.
 - Konvertierungs- und Rundungsfehler bei Festpunktdarstellungen
- ⇒ **approximative (angenäherte) Darstellung mit möglichst vielen "signifikanten Stellen" in der Zahlendarstellung**
- ⇒ **dynamische Komma-Verwaltung als Bestandteil der Zahlendarstellung**



Gleitkommadarstellung

- Die Gleitkommadarstellung (auch Gleitpunktdarstellung, *floating point*) einer Zahl x (primär $x \in \mathbb{R}$) besitzt die Form

$$x = m * B^e$$

- B ist die *Basis*, typisch $B = 2, 10$ oder 16 .
- e ist der ganzzahlige *Exponent* (bestimmt die Größenordnung der Zahl)
- m ist eine vorzeichenbehaftete Festkommazahl und heißt die *Mantisse* von x (Vorzeichen von m ist unabhängig vom Vorzeichen von e).
- m und e werden zur Basis B dargestellt.
- Die Darstellung ist eindeutig, wenn die Mantisse *normalisiert* ist, d.h. $1/B \leq |m| < 1$ oder $1 \leq |m| < B$ für $m \neq 0$ gilt (Konventionssache).



Gleitkommadarstellung (2)

- **Beispiel:**
 $-12,25_{10} = -1100,01_2 = -0,110001_2 * 2^4 = -1,10001_2 * 2^3$
- **Prinzipielle Repräsentierung einer Gleitpunktzahl in einem Maschinenwort**



Def

- Der Exponent e wird i.d.R. in Excess-k-Darstellung (vgl. 3.3.3) gespeichert und dann auch als **Charakteristik** (ch) bezeichnet.



Gleitpunktarithmetik

- Die Gleitpunktarithmetik für die Grundrechenarten zweier Operanden $x = m * B^e$ und $x' = m' * B^{e'}$ ist gegeben durch

$$x + x' = (m + m' B^{e'-e}) * B^e$$

$$x - x' = (m - m' B^{e'-e}) * B^e$$

$$x * x' = (m * m') * B^{e+e'}$$

$$x / x' = (m / m') * B^{e-e'}$$

Verschiebung der Mantissen
zur Anpassung der Exponenten

Anschließend normalisieren!

- Bei verschiedenen Exponenten (Charakteristika) wird eine automatische Anpassung des kleineren Exponenten durchgeführt. Hierbei kann es zu Rundungsfehlern und Auslöschungen kommen.

- Beispiel

(Dezimalsystem, 4 Mantissenstellen):

$$\begin{array}{r} 0,5400 \quad *10^4 \\ + 0,2000 \quad *10^{-1} \\ \hline 0,5400 \quad *10^4 \\ + 0,000002000 *10^4 \\ \hline 0,5400 \quad *10^4 \end{array}$$

Verschiebung der Mantissen
zur Anpassung der Exponenten



Gleitpunktformat nach IEEE 754

- IEEE: Institute of Electrical and Electronics Engineers
- Gleitpunktformate nach IEEE 754 werden von fast allen heutigen Prozessoren durch spezielle Gleitpunktrechnwerke unterstützt.
- Daneben gibt es weitere Formate, z.B. Intel x86 Extended Real (80 Bit)
- Festlegungen:
 - Basis $B = 2$
 - für zwei Maschinenwortlängen ausgelegt: 32 Bit, 64 Bit
 - entsprechend C `float` und `double` bzw. `long double`
 - Vorzeichen: + dargestellt durch 0, - durch 1

Wortlänge	32 Bit	64 Bit
Charakteristik	8 Bit	11 Bit
Excess-k	$k = 127$	$k = 1023$
Mantisse	23 Bit	52 Bit
norm. Betragsbereich	$\approx 10^{-38} - 10^{38}$	$\approx 10^{-308} - 10^{308}$
gültige Dezimalstellen	ca. 6	ca. 16



Besonderheiten

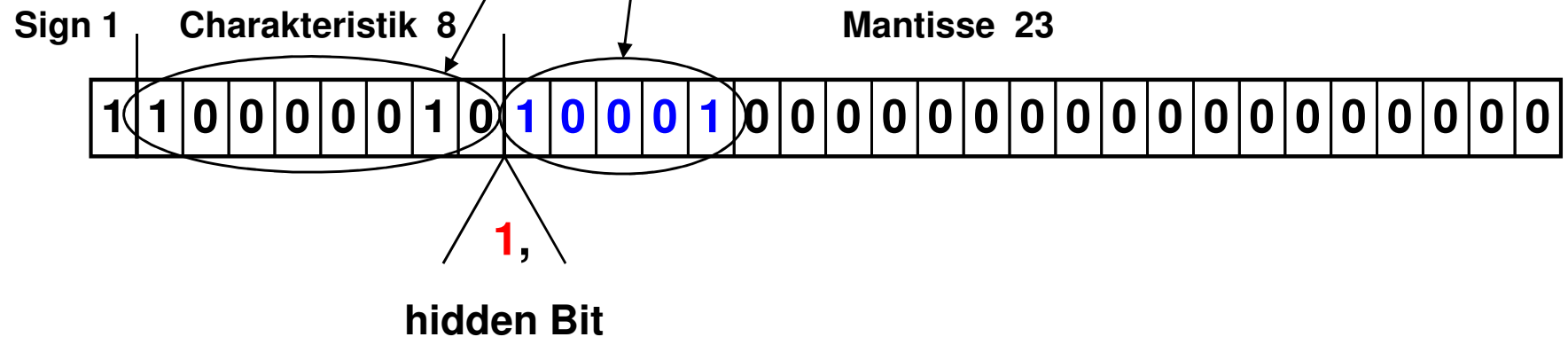
- IEEE-Gleitpunktdarstellung verwendet *hidden Bit*, d.h. das erste Bit einer normalisierten Mantisse, das im Falle $B=2$ immer 1 ist, wird nicht gespeichert, aber bei allen Operationen als vorhanden angenommen (Gewinn einer zusätzlichen Binärstelle).
- Der größte und der kleinste Exponent ($\text{Excess-127} \Rightarrow \text{Charakteristik } ch=0 \text{ und } ch=255_{10}$) werden für *Sonderfälle* verwendet.
- Überblick:

ch	m	Wert	Bemerkung
0	0	$(-1)^s * 0$	± 0
0	$\neq 0$	$(-1)^s * 2^{-k+1} * (0.m)$	nicht-normalisiert, Zahlen mit Betrag kleiner als normalisiert darstellbar
$0 < ch < 2k+1$	m beliebig	$(-1)^s * 2^{ch-k} * (1.m)$	Normalfall
$ch = 2k+1$	0	$(-1)^s * \infty, \text{INF}, -\text{INF}$	$\pm \infty$
$ch = 2k+1$	$\neq 0$	NaN	Not a Number: unbestimmter oder unzulässiger Wert



Beispiel

- Darstellung von $-12,25_{10}$ im IEEE 754-Format für 32 Bit
- $-12,25_{10} = -1100,01_2 = -1,10001_2 * 2^3$
- $ch = 3_{10} + 127_{10} = 130_{10} = 10000010_2$





Anmerkungen

- Die Verwendung eines Gleitpunktformates führt zu einer nicht-äquidistanten Zahlenverteilung der darstellbaren Zahlen, d.h. der Abstand zwischen benachbarten darstellbaren Zahlen ist unterschiedlich.
- Das Problem der nur schwer abschätzbaren Rechengenauigkeit bleibt auch bei Gleitpunktzahlen erhalten.
- Bei bestimmten Anwendungen kann daher sinnvoll sein:
 - Übergang zu einer Software-Lösung für die exakte Arithmetik z.B. sehr großer Zahlen (dargestellt in Folgen von Maschinenwörtern)
 - Übergang zur symbolischen Formelmanipulation (Computer-Algebra) und Einsetzen von numerischen Werten erst zum Schluss (z.B. Maple, Mathematika).



3.4 Zeichenketten

- **Ziel: Repräsentierung von Wörtern über einem ungeordneten Zeichenvorrat oder einem Alphabet von Schriftzeichen.**
- **Ein Schriftzeichen-Alphabet umfasst i.d.R.**
 - **Buchstaben (Groß- und Klein-Buchstaben)**
 - **Ziffern**
 - **druckbare Sonderzeichen**
 - **evtl. sogenannte Steuerzeichen, die durch ein verarbeitendes Gerät interpretiert werden.**
- **Man spricht daher auch von **alphanumerischen** Codes.**



3.4 Zeichenketten (2)

- **Die wichtigsten alphanumerischen Codes, die im weiteren vorgestellt werden, sind:**
 - CCITT-Code No. 2 (historisch)
 - ASCII, ISO 646
 - erweiterter ASCII-Code (PC8)
 - ISO 8859-1 bis -15
 - EBCDIC
 - UCS bzw. UNICODE bzw. ISO 10646,
 - incl. UTF-16 und UTF-8 Codierung



Operationen auf Zeichenketten

- **Zeichenketten werden in Rechensystemen in Bytefolgen gespeichert.**
- **Maschinenoperationen sind häufig auf Zeichenketten einer bestimmten maximalen Länge beschränkt (z.B. 255 Zeichen).**
 - **Vorsicht, falls Anzahl Zeichen \neq Anzahl Bytes!**
- **Neben Operationen zum Kopieren von Zeichenketten (allgemeiner von Bytefolgen) ist das Vergleichen von Zeichenketten in Hinblick auf die lexikographische Ordnung (vgl. Kap. 2.4) besonders wichtig.**



3.4.1 CCITT-Code No. 2

- **CCITT:**
Comité Consultatif International Telegraphique et Telephonique, jetzt International Telecommunications Union (ITU) genannt
- von D. Murray ca. 1901 entwickelter **Fernschreibcode**
- **5-Bit Code** ($2^5=32$ Codewörter)
(entsprechend 5 Spuren eines Lochstreifens plus Taktspur)
- Steuerzeichen zur Umschaltung zwischen Buchstabenmodus und Ziffern/Sonderzeichen-Modus verdoppelt den Vorrat an Codewörtern
- Code optimiert zur Minimierung des mechanischen Verschleißes
 - Häufig benutzte Buchstaben wie E und T bewegen wenige Teile

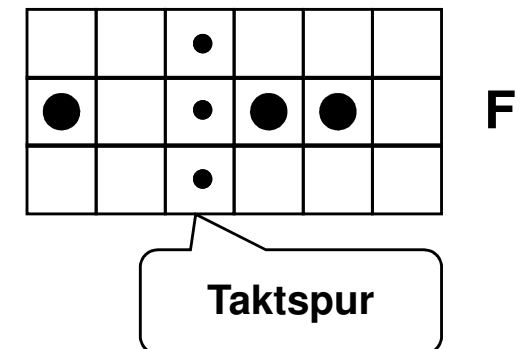


Code-Tabelle CCITT-2

Code-Nr.	Dual-code	Buchstabe	Ziffer
1	11000	A	-
2	10011	B	?
3	01110	C	:
4	10010	D	WAY
5	10000	E	3
6	10110	F	(NA)
7	01011	G	(NA)
8	00101	H	(NA)
9	01100	I	8
10	11010	J	<i>bell</i>
11	11110	K	(
12	01001	L)
13	00111	M	.
14	00110	N	,
15	00011	O	9
16	01101	P	0

Code-Nr.	Dual-code	Buchstabe	Ziffer
17	11101	Q	1
18	01010	R	4
19	10100	S	'
20	00001	T	5
21	11100	U	7
22	01111	V	=
23	11001	W	2
24	10111	X	/
25	10101	Y	6
26	10001	Z	+
27	00010	CR	
28	01000	LF	
29	11111	LS	
30	11011	FS	
31	00100	<i>space</i>	
32	00000	<i>(unused)</i>	

WAY	Who Are You? (Wer da?)
<i>bell</i>	Klingel
CR	Carriage Return (Wagenrücklauf)
LF	Line Feed (Zeilenvorschub)
LS	Letter Shift (Buchstabenumsch.)
FS	Figure Shift (Ziffernumschaltung)
<i>space</i>	Zwischenraum
(NA)	Not Assigned (nicht definiert, reserviert)
<i>(unused)</i>	nicht benutzt





3.4.2 ASCII-Code

- **ASCII: American Standard Code for Information Interchange (1963), ANSI X3.4 (1968)**
- **ationale amerikanische Version, auch als US-ASCII bezeichnet**
- **entspricht CCITT-Code No. 5**
- **keine Umlaute, diese in oft nicht standardisierten Varianten statt einiger Sonderzeichen.**
 - **Standardisierte Varianten: ISO 646 (1972), US-ASCII = ISO 646-US**
 - **z.B. ISO 646-de: @, [, \,], {, |, }, ~ → \$, Ä, Ö, Ü, ä, ö, ü, ß**
 - **sehr verbreitet zur Codierung von Zeichenketten in Rechensystemen**
- **7-Bit Code ($2^7=128$ Codewörter)**
- **Speicherung eines Zeichens in einem Byte**
- **Steuerzeichen (*Control Codes*) in den Positionen 0-31 und 127**
 - **Daher: Steuerungs- bzw. Control-Taste der Keyboards!**



Code-Tabelle US-ASCII

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
00	NUL	10	DLE	20	SP	30	0
01	SOH	11	DC1	21	!	31	1
02	STX	12	DC2	22	"	32	2
03	ETX	13	DC3	23	#	33	3
04	EOT	14	DC4	24	\$	34	4
05	ENQ	15	NAK	25	%	35	5
06	ACK	16	SYN	26	&	36	6
07	BEL	17	ETB	27	'	37	7
08	BS	18	CAN	28	(38	8
09	HT	19	EM	29)	39	9
0A	LF	1A	SUB	2A	*	3A	:
0B	VT	1B	ESC	2B	+	3B	;
0C	FF	1C	FS	2C	,	3C	<
0D	CR	1D	GS	2D	-	3D	=
0E	SO	1E	RS	2E	.	3E	>
0F	SI	1F	US	2F	/	3F	?

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
40	@	50	P	60	`	70	p
41	A	51	Q	61	a	71	q
42	B	52	R	62	b	72	r
43	C	53	S	63	c	73	s
44	D	54	T	64	d	74	t
45	E	55	U	65	e	75	u
46	F	56	V	66	f	76	v
47	G	57	W	67	g	77	w
48	H	58	X	68	h	78	x
49	I	59	Y	69	i	79	y
4A	J	5A	Z	6A	j	7A	z
4B	K	5B	[6B	k	7B	{
4C	L	5C	\	6C	l	7C	
4D	M	5D]	6D	m	7D	}
4E	N	5E	^	6E	n	7E	~
4F	O	5F	_	6F	o	7F	DEL

HEX	ASCII	Bedeutung
02	STX	Textanfang (<i>start of text</i>)
03	ETX	Textende (<i>end of text</i>)
07	BEL	Klingel (<i>bell</i>)
09	HT	Tabulator (<i>horizontal tabulator</i>)
0A	LF	Zeilenvorschub (<i>line feed</i>)
0C	FF	Seitenvorschub (<i>form feed</i>)
0D	CR	Wagenrücklauf (<i>carriage return</i>)
1B	ESC	Umschaltung (<i>Escape</i>)

Ordnung:

Es gilt insbesondere

Leerzeichen < 0 < 1 < ... < 9 < A < B < ... < Z < a < b < ... < z

Einige wichtige Steuer-Codes

Demo!

Eingabebeispiele:

Strg-D=EOT (cat!), Strg-G = BEL, Strg-J=LF



3.4.2 ASCII-Code

- Zum Vergleich: ISO-646-DE

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
40	S	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	Ä	Ö	Ü	^
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	ä	ö	ü	ß

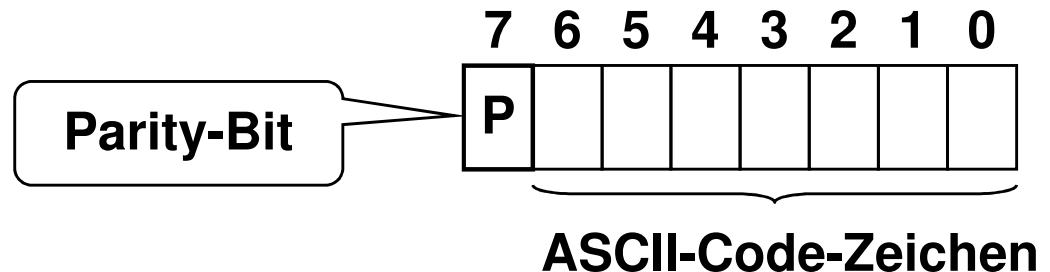
- Ordnung:
 - Problematisch bei Umlauten

Bei Induzierung der lexikographischen Ordnung aus der Ordnung der Bitmuster (etwa: $A < B$, weil $01000001 < 01000010$) werden die Umlaute hinter den letzten Buchstaben sortiert: $z < ä$ statt $a < ä < b$.



Paritätsprüfung

- Das zur Speicherung eines Zeichens in einem Byte nicht benötigte Bit kann zur Fehlerkontrolle (Paritätsprüfung) eingesetzt werden



- Unterscheidung
 - gerade Parität (*even parity*):
P=0, wenn Anzahl der "1" im Code bereits gerade ist,
P=1 sonst (damit insgesamt wieder gerade Anzahl)
 - ungerade Parität (*odd parity*): umgekehrt
- Paritätsprüfung (*parity check*)
 - Erkennung von 1-Bit-Übertragungsfehlern (alle ungeraden Anzahlen)
 - Berechnetes Parity-Bit wird mit übertragen
 - Empfänger berechnet seinerseits das Parity-Bit und vergleicht es mit dem empfangenen: bei Ungleichheit Fehler erkannt.



3.4.3 Erweiterter ASCII-Code (PC8)

- auf 8 Bit erweiterter ASCII-Code
- Zeichen 0...127 entsprechen dem 7-Bit ASCII-Code
- das 8. Bit wird in die Codierung einbezogen, um die Anzahl der codierbaren Zeichen auf $2^8=256$ zu erhöhen.
- insbesondere genutzt für länderspezifische Sonderzeichen und semigraphische Symbole
- gebräuchlicher Zeichensatz auf PCs
 - IBM Codepage 437
- Varianten:
 - 850 (mehrsprachig, Schwerpunkt Europa)
 - Einige Rahmen ersetzt durch weitere buchstabenartige Zeichen
 - 858 (einschließlich €-Symbol)
 - 865 (Nordisch), 866 (Russisch/Kyrillisch), usw.
 - 1252 (Microsoft, Obermenge von ISO 8859-1, s.u.)



Code-Tabelle PC8

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`	128	80	Š	160	A0	á	192	C0	Ł	224	E0	α	256	FF				
^A	1	01	␣	SOH	33	21	!	65	41	A	97	61	a	129	81	š	161	A1	â	193	C1	ł	225	E1	β	257					
^B	2	02	␣	SIX	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	ł	226	E2	Γ	258					
^C	3	03	♥	EIX	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ô	195	C3	ł	227	E3	Π	259					
^D	4	04	♦	EOI	36	24	\$	68	44	D	100	64	d	132	84	â	164	A4	ñ	196	C4	ł	228	E4	Σ	260					
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e	133	85	â	165	A5	ñ	197	C5	ł	229	E5	σ	261					
^F	6	06	♣	ACK	38	26	&	70	46	F	102	66	f	134	86	â	166	A6	ñ	198	C6	ł	230	E6	ρ	262					
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g	135	87	â	167	A7	ñ	199	C7	ł	231	E7	τ	263					
^H	8	08	␣	BS	40	28	(72	48	H	104	68	h	136	88	â	168	A8	ñ	200	C8	ł	232	E8	ϑ	264					
^I	9	09	o	HI	41	29)	73	49	I	105	69	i	137	89	â	169	A9	ñ	201	C9	ł	233	E9	θ	265					
^J	10	0A	␣	LF	42	2A	*	74	4A	J	106	6A	j	138	8A	â	170	AA	ñ	202	CA	ł	234	EA	Ω	266					
^K	11	0B	♣	VI	43	2B	+	75	4B	K	107	6B	k	139	8B	â	171	AB	ñ	203	CB	ł	235	EB	δ	267					
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l	140	8C	â	172	AC	ñ	204	CC	ł	236	EC	⊙	268					
^M	13	0D	♣	CR	45	2D	-	77	4D	M	109	6D	m	141	8D	â	173	AD	ñ	205	CD	ł	237	ED	ϑ	269					
^N	14	0E	♣	SO	46	2E	.	78	4E	N	110	6E	n	142	8E	â	174	AE	ñ	206	CE	ł	238	EE	€	270					
^O	15	0F	♣	SI	47	2F	/	79	4F	O	111	6F	o	143	8F	â	175	AF	ñ	207	CF	ł	239	EF	∩	271					
^P	16	10	♣	SLE	48	30	0	80	50	P	112	70	p	144	90	â	176	B0	ñ	208	D0	ł	240	F0	≡	272					
^Q	17	11	♣	CS1	49	31	1	81	51	Q	113	71	q	145	91	â	177	B1	ñ	209	D1	ł	241	F1	+	273					
^R	18	12	♣	DC2	50	32	2	82	52	R	114	72	r	146	92	â	178	B2	ñ	210	D2	ł	242	F2	>	274					
^S	19	13	!!	DC3	51	33	3	83	53	S	115	73	s	147	93	â	179	B3	ñ	211	D3	ł	243	F3	<	275					
^T	20	14	␣	DC4	52	34	4	84	54	T	116	74	t	148	94	â	180	B4	ñ	212	D4	ł	244	F4	⌋	276					
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u	149	95	â	181	B5	ñ	213	D5	ł	245	F5	⌋	277					
^V	22	16	♣	SYN	54	36	6	86	56	V	118	76	v	150	96	â	182	B6	ñ	214	D6	ł	246	F6	÷	278					
^W	23	17	♣	EIB	55	37	7	87	57	W	119	77	w	151	97	â	183	B7	ñ	215	D7	ł	247	F7	≈	279					
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x	152	98	â	184	B8	ñ	216	D8	ł	248	F8	°	280					
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y	153	99	â	185	B9	ñ	217	D9	ł	249	F9	•	281					
^Z	26	1A	→	SIB	58	3A	:	90	5A	Z	122	7A	z	154	9A	â	186	BA	ñ	218	DA	ł	250	FA	·	282					
^[27	1B	+	ESC	59	3B	;	91	5B	[123	7B	{	155	9B	â	187	BB	ñ	219	DB	ł	251	FB	⌋	283					
^\	28	1C	␣	FS	60	3C	<	92	5C	\	124	7C		156	9C	â	188	BC	ñ	220	DC	ł	252	FC	⌋	284					
^]	29	1D	+	GS	61	3D	=	93	5D]	125	7D	}	157	9D	â	189	BD	ñ	221	DD	ł	253	FD	z	285					
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~	158	9E	â	190	BE	ñ	222	DE	ł	254	FE	■	286					
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	Δ†	159	9F	â	191	BF	ñ	223	DF	ł	255	FF		287					

† ASCII code 127 has the code DEL. Under MS-DCS, this code has the same effect as ASCII 8 (BS).
The DEL code can be generated by the CTRL+BKSP key.



3.4.4 EBCDIC

- **EBCDIC: Extended Binary Coded Decimal Interchange Code**
- **8-Bit-Code**
- **Als Erweiterung des BCD-Codes zur Anwendung in IBM System/360-Rechnern von IBM entwickelt und von anderen Herstellern übernommen**
- **Noch immer aktuell:**
 - Heute noch in Großrechnern (Mainframes) angewendet
 - In 2004 Anlass für die Verabschiedung von XML 1.1



3.4.5 ISO 8859

- ISO: International Organization for Standardization, www.iso.org
- ISO 8859:
 - Familie von 8-Bit-Codes
 - Standardisierung von IBMs Codepage-Ansatz
- ISO 8859-1 (ISO Latin-1)
 - Für West- und Mitteleuropa ausgelegt
 - Nachfolger: ISO 8859-15 (incl. €-Zeichen)

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	í	φ	£	℥	¥	ı	š	..	©	≡	«	¬	—	®	—
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ

Quelle:
<http://czyborra.com>



ISO 8859 (2)

- **ISO 8859-15 (ISO Latin-9)**
 - Nachfolger von ISO 8859-1, insbesondere incl. €-Zeichen
 - War bis vor wenigen Jahren der wichtigste Zeichensatz hierzulande
 - Z.B. Grundlage für Einstellungen Ihres E-Mail-Clients
- **Unterschiede zwischen ISO 8859-1 und 8859-15:**

Position	0xA4	0xA6	0xA8	0xB4	0xB8	0xBC	0xBD	0xBE
8859-1	¤	¦	¨	´	¸	¼	½	¾
8859-15	€	Š	š	Ž	ž	Œ	œ	Ÿ

Quelle:
Wikipedia.de



ISO 8859 (3)

- **Die Code-Tabellen von ISO 8859:**
 - **ISO 8859-1 (ISO Latin 1), für West- und Mitteleuropa**
 - **ISO 8859-2 (ISO Latin 2), für Mittel- und Osteuropa**
 - **ISO 8859-4 (ISO Latin 4), für die baltische Sprachen, Grönländisch, Lappisch (Sami)**
 - **ISO 8859-5 (Kyrillisch)**
 - **ISO 8859-6 (Arabisch)**
 - **ISO 8859-7 (Griechisch)**
 - **ISO 8859-8 (Hebräisch)**
 - **ISO 8859-9 (ISO Latin 5), für Türkisch**
 - **ISO 8859-10 (ISO Latin 6), für alle Nordischen Sprachen**
 - **Inoffiziell: -11 (Thai), -12 (reserviert), -13 (Baltische Staaten), -14 (Keltische Sprachen)**
 - **ISO 8859-15 (ersetzt ISO 8859-1, ergänzt u.a. das Euro-Zeichen)**
-



3.4.6 Unicode

- Der Unicode-Zeichensatz ist ein relativ neues, international standardisiertes Code-System, das alle Schriftzeichen der verbreiteten internationalen Schriften sowie historischer Schriftarten enthält, insbesondere auch Chinesisch/Japanisch/Koreanisch (CJK).
- Standardisierung durch Unicode-Konsortium (<http://www.unicode.org>)
- Unicode besitzt große Bedeutung für die Internationalisierung von Programmen.
- Unicode ist konform zur internationalen Norm ISO/IEC 10646 (Universal Character Set, UCS).
- Unicode definiert *kein* äußeres Erscheinungsbild (Glyph) für Zeichen, wie dies Fonts tun (z.B. Arial, Helvetica).

Auch Klingonisch?

Das Unicode-Konsortium weigerte sich hartnäckig...



Unicode-Zeichen

- Jedes Zeichen besitzt eine 32(31)-Bit-Zeichenummer.
- 20.06.17: Unicode 10.0.0, 136690 Zeichen (Unicode 2.0: 38885)
- Die wichtigsten Zeichen passen in die ersten 2^{16} „Fächer“
 - *Basic Multilingual Plane* (BMP), hier genügt eine 2-Byte-Codierung.
- Über 860000 reservierte, noch ungenutzte Ergänzungen
 - 6400 Plätze der BMP für private Anwendungen reserviert
 - 131068 für private Anwendungen in anderen Ebenen.
- Jedes Zeichen wird eindeutig über seine Nummer oder seine (ebenfalls standardisierte) textuelle Beschreibung identifiziert
 - gebräuchlichste Schreibweise (für Zeichen aus der BMP):
U+xxxx, wobei xxxx eine vierstellige hexadezimale Zahl ist.
- Beispiele:
 - U+0041 "LATIN CAPITAL LETTER A"
 - U+20AC "EURO SIGN" (Euro-Währungszeichen)

z.B. für die StarTrek-Fans: Codierung der Klingonischen Glyphen im „privaten Bereich“ E000-F8FF



Zeichenbereiche

- Die Unicode-“Codepoints“ sind in Zeichenbereiche (insb. *Scripts*) aufgeteilt. Diese Bereiche spiegeln jeweils eine bestimmte Schriftkultur oder einen Satz von Sonderzeichen wider.
- Unicode ist abwärtskompatibel:
 - Der Bereich U+0000 bis U+007F "C0 Controls and Basic Latin" entspricht genau dem ASCII-Standard.
 - Der Bereich U+0080 bis U+00FF "C1 Controls and Latin-1 Supplement" ist identisch mit ISO 8859-1.
- Weitere Beispiele für Zeichenbereiche:
 - U+0370 bis U+03FF Griechisch
 - U+20A0 bis U+20CF Währungssymbole
- Zeichenbereiche sind unterschiedlich groß
 - ASCII: 128 Zeichen, Währungssymbole: 48 Zeichen
 - CJK-Block enthält Tausende von Zeichen (U+4E00 ... U+BFFF)



Zeichenrepräsentierung

- Der Unicode-Standard sieht verschiedene Codierungen des Zeichensatzes vor, entsprechend den ISO 10646
Transformationsformaten UTF-8 und UTF-16, die ohne Informationsverlust ineinander überführt werden können.
- UTF-16:
 - Speicherung jedes Unicode-Zeichens der BMP in 2 Bytes (feste Codelänge) mit einem Inhalt entsprechend der Zeichennummer
 - Beispiel: U+0041 \Rightarrow 0041₁₆
 - **Byte Order Mark (BOM)** U+FEFF = erstes Zeichen einer UTF-16 Datei
 - Zur automatischen Erkennung Little-Endian / Big-Endian, vgl. Kap. 3.3.3
- UTF-8 (u.a. RFC 3629):
 - Variable Codelänge: 1, 2, ..., 6 Bytes pro Zeichen!
 - Erlaubt kompaktere Darstellungen (vgl. Kap. 4).
 - Abwärtskompatibel zu US-ASCII: Die Codierung für den Zeichenbereich U+0000 bis U+007F entspricht genau dem ASCII-Code!

Demo!




UTF-8

- **UTF-8 Codierung:**

Unicode-Zeichenbereich	UTF-8 Codierung (Bytefolge)
U+00000000 - U+0000007F	0xxxxxxx ₂ Anzahl Folgebytes
U+00000080 - U+000007FF	110xxxxx ₂ , 10xxxxxx ₂
U+00000800 - U+0000FFFF	1110xxxx ₂ , 10xxxxxx ₂ , 10xxxxxx ₂
U+00010000 - U+001FFFFF	11110xxx ₂ , (10xxxxxx ₂) ₃ (3 Folgebytes)
U+00200000 - U+03FFFFFF	111110xx ₂ , (10xxxxxx ₂) ₄ (4 Folgebytes)
U+04000000 - U+7FFFFFFF	1111110x ₂ , (10xxxxxx ₂) ₅ (5 Folgebytes)

- Die Stellen x...x werden von rechts mit den Bits des Unicode-Werts befüllt
- 1 bis 6 Oktetts pro Unicode-Zeichen (31 Bit), niemals FE₁₆ oder FF₁₆.
- Stets klar, ob Folgebyte vorliegt und wie viele Folgebytes notwendig!

- **Beispiel:**

- A U+41 → 01000001₂ = 41₁₆ unverändert (ASCII !)
- Ä U+C4 → (11000011₂ , 10000100₂) = (C3₁₆ , 84₁₆)
-  U+1D11E → (F0₁₆ , 9D₁₆ , 84₁₆ , 9E₁₆)

Demo!



Unicode: Neue Qualitäten

- **Mehrdeutigkeit der Darstellung einiger Zeichen**
(→ *combining characters*)
 - „ü“: Direkt „ü“ (U+00FC) oder kombiniert „“ + u“ (U+0308, U+0075)
 - Folge: Vergleichbarkeit von Strings erfordert Normalisierung
- **Große Verwechslungsgefahr bei einigen Zeichen**
 - Großes griech. Omega (Ω, U+03A9) vs. Zeichen „Ohm“ für elektr. Widerstand (Ω, U+2126), o (U+006F) vs. o (kl. Omicron, U+03BF)
 - Folge: Vorsicht bei Unicode-Zeichen in URLs – Phishing-Risiko!
- **Variable Schreibrichtung**
 - z.B. „rechts nach links“ bei arabischen oder hebräischen Texten
- **Die Kunst der Definition einer lexikographischen Ordnung**
 - UCA (*Unicode Collation Algorithm*, <http://www.unicode.org/unicode/reports/tr10/>)
 - Von Lokalisierung des Rechners abhängig, z.B. „SE-se“ vs. „DE-de“
 - Ordnung nicht von *code points* (Binärdarstellung) induziert!



Unicode im Alltag

- **Unicode begegnet Ihnen bereits heute**
 - **In vielen Produkten von Microsoft**
 - Siehe Demo (Texteditor)
 - Exportformat der Registry
 - Word, PowerPoint u.v.a.m.
 - **Analog auf dem Apple Macintosh**
 - **In Quellcodes moderner Programmier- und Auszeichnungssprachen**
 - Java, Ruby
 - XML, HTML
- **Bei Bedarf: Umwandlung zwischen Zeichensätzen**
 - **Bewährtes Werkzeug auf der Unix-Kommandozeile:**
 - iconv
 - **Aber: Vorsicht vor „unmöglichen“ Wünschen**
 - Die meisten Unicode-Zeichen haben keine Entsprechung etwa in ISO 8859

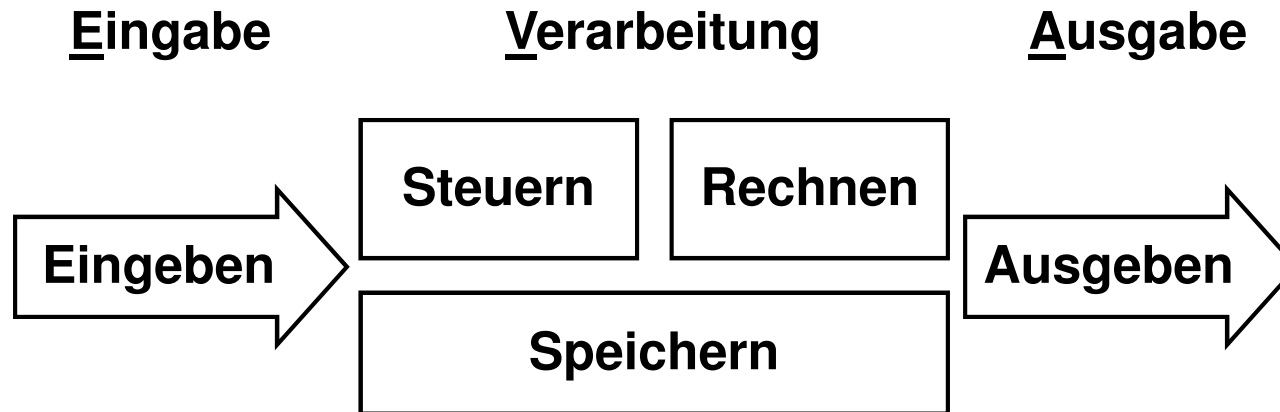
Demo: Arial Unicode MS

Demo!



3.5 Ein- / Ausgabe

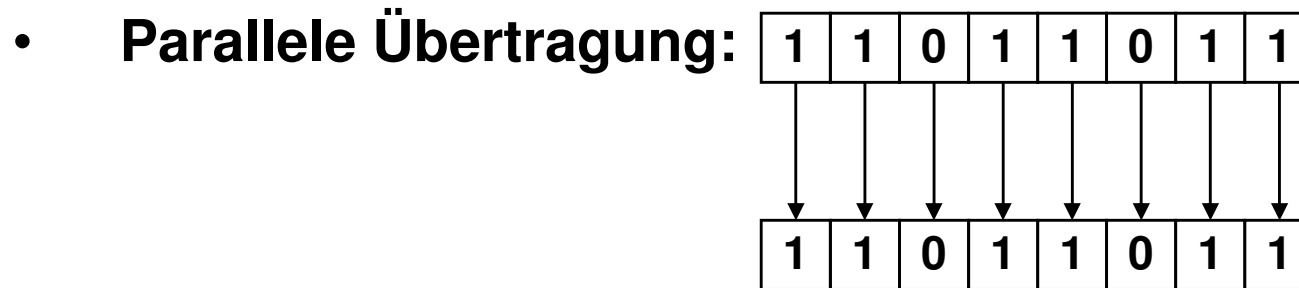
- **Modell eines Rechensystems (vgl. Kap. 1)**



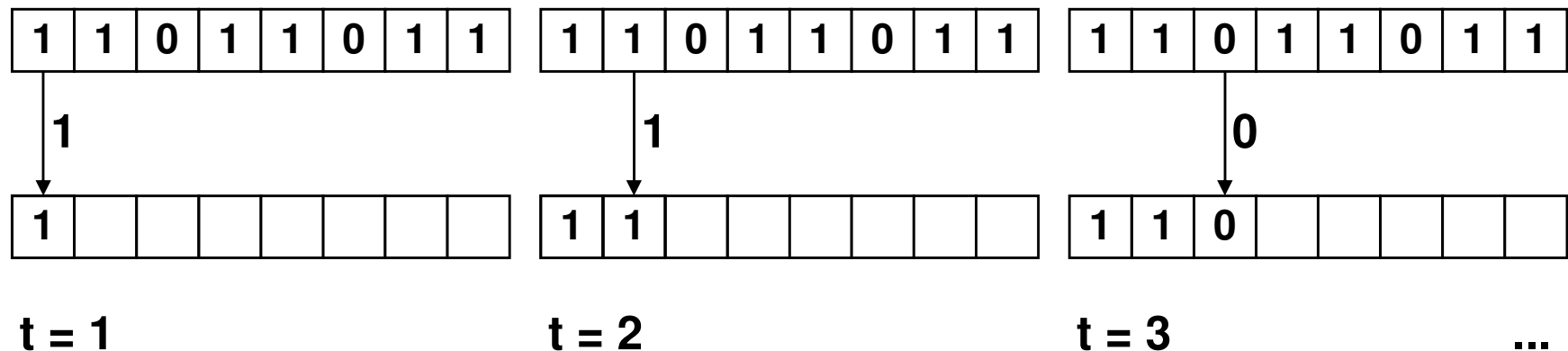
- **Die Verarbeitung geschieht auf Basis digitaler Informationen, Ein- und Ausgabe von/in die Umgebung des Rechensystems kann auf digitalen Signalen oder analogen Signalen basieren.**

✱ Parallele und sequenzielle Übertragung

- Die Übertragung einer Signalmenge zwischen den Komponenten eines Rechensystems bzw. von/zur Umgebung des Rechensystems kann *parallel* im Raum oder *sequenziell* in der Zeit geschehen.



- Sequenzielle Übertragung:





Analoge und digitale Signale

Def

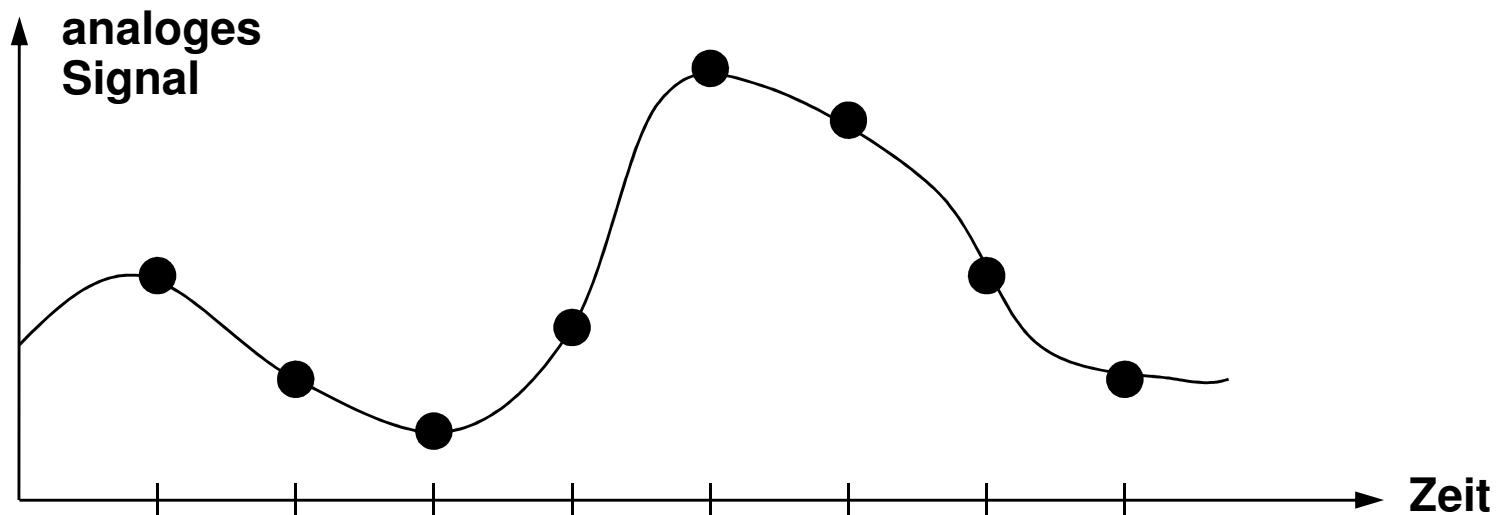
- Der zeitlich veränderliche Verlauf einer physikalischen Größe heißt **Signal**.
- Beispiele:
 - Spannungssignal, akustisches Signal, optisches Signal.
- Ein Signal heißt **analog** oder *kontinuierlich*, wenn es kontinuierliche Werte entsprechend einem Intervall aus der Menge der reellen Zahlen annehmen kann (unendlich viele Werte).
- Beispiele:
 - **Spannungssignale** als Sensorwerte von physikalischen Größen wie Temperatur, Lautstärke, **Helligkeit**, Winkel, ...
- Ein Signal heißt **digital**, wenn es nur endlich viele Werte annehmen kann.
- Beispiel: Schalter (offen oder geschlossen)



Rasterung/Abtastung

Def

• **Rasterung** oder **Abtastung**: der Wert eines (analogen oder digitalen) Signals wird nur zu einzelnen Zeitpunkten (z.B. mit festem Intervall) oder einzelnen Ortspunkten bestimmt. Die Größe wird dadurch **diskretisiert**. Der ermittelte Wert kann z.B. einem Funktionswert im betrachteten Intervall oder einem Mittelwert entsprechen.

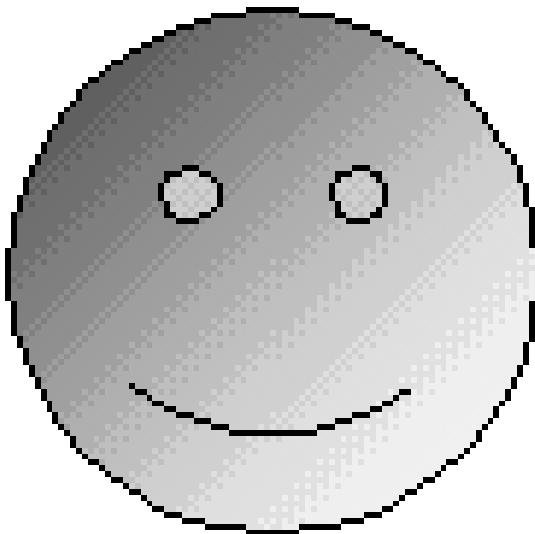


- Anmerkung: Die Bedingungen, die erfüllt sein müssen, damit die Abtastung zu keinem Informationsverlust führt, werden durch das sogenannte **Abtasttheorem** von Shannon definiert (vgl. Vorlesung Informations- und Systemtheorie).



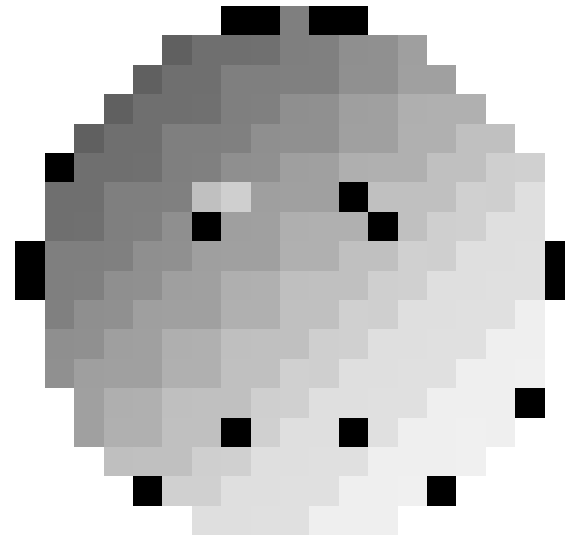
Rasterung/Abtastung (2)

- Beispiel: 2-dimensionale Rasterung (Ortspunkte)



Original

in Wirklichkeit auch
schon (feiner) gerastert



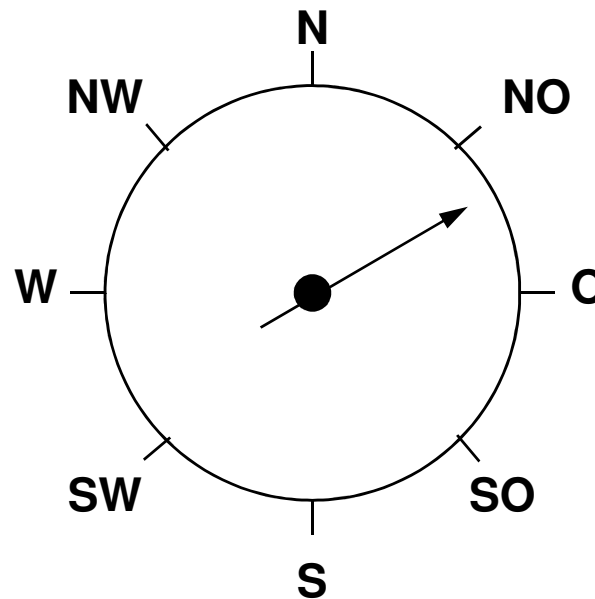
Rasterung



Quantelung

Def

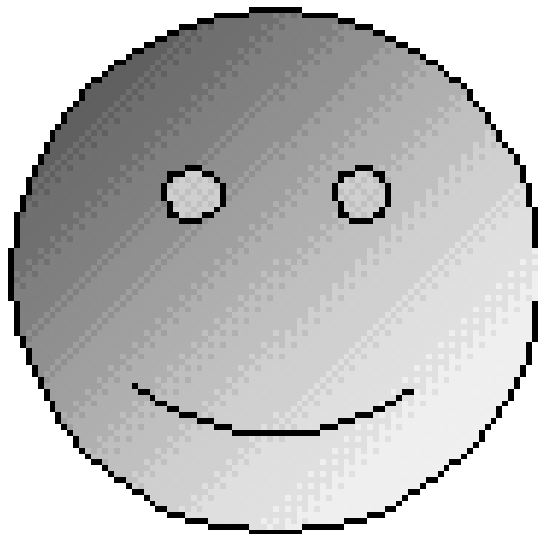
- Unter **Quantelung** versteht man die Abbildung eines analogen Signals in ein digitales Signal (Diskretisierung des Werts).
- Beispiel 1:
 - Windrichtung: Winkelbereich zwischen 0 und 360 Grad wird in 8 gleiche Abschnitte unterteilt, die mit {N, NO, O, SO, S, SW, W, NW} bezeichnet werden. Es wird der nächstgelegene Wert zugeordnet.



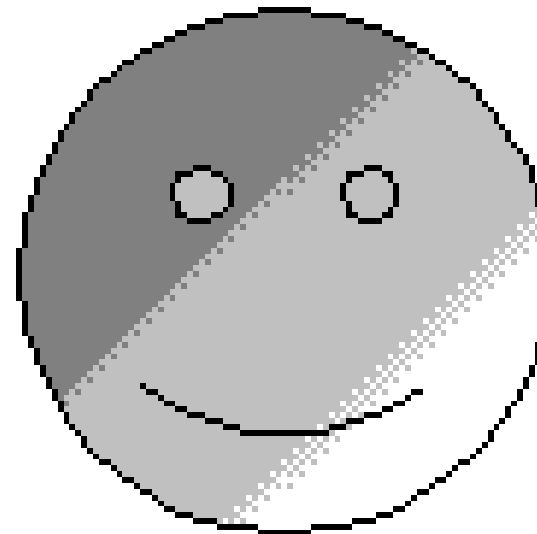


Quantelung (2)

- **Beispiel 2: 2-dimensionale Quantelung (Ortspunkte)**



Original
in Wirklichkeit auch
schon gequantelt



Quantelung
auf 4 Grauwerte



Quantelung (3)

- Die technische Realisierung einer Quantelung geschieht häufig durch sogenannte *Analog/Digital-Wandler*, die einen vorgegebenen Spannungsbereich in n unterschiedliche Werte abbilden (z.B. $n=1024$).
- Die Kombination aus Abtastung (in Zeit bzw. Raum) und Quantelung (der Amplitude) wird auch als *Pulse-Code-Modulation (PCM)* bezeichnet.
- Beispiel:

Audio-CD: Abtastung 44.100 Mal/sec (44.1 kHz), Wertebereich 2^{16}