HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Technische Grundlagen der Informatik

Darstellung von Zahlen und Zeichen

Prof. Dr.-Ing. Benjamin Weinert

Gliederung

- Zahlen
  - Einfache Zahlendarstellungen
  - Stellenwertsysteme
  - Rationale Zahlen
  - Umwandlungen zwischen Zahlensystemen
- Computerzahlen
  - Vorzeichenlose, ganze Zahlen
  - Vorzeichenbehaftete ganze Zahlen
  - Vorzeichenbehaftete rationale Zahlen
- Zeichen
  - ASCII-Code
  - ISO 8859
  - Unicode

Zahlen

- Entstanden ursprünglich aus den Aufgabenstellungen
  - des Zählens
    - “Ein Bauer besitzt 23 Kühe.”
    - **Kardinalzahlen**
  - des Ordnens
    - “der Erste, der Zweite, ...”
    - **Ordinalzahlen**

- Früheste Darstellung: Strichsysteme
  - Zahlen werden durch die Wiederholung eines speziellen Zeichens dargestellt
  - Auch heute noch gebräuchlich: Liste Kaffeekasse
  - Praktisch bei kleinen Zahlen, für große Zahlen nicht anwendbar
  - Addition ergibt sich natürlich („Additionssysteme“)

Das römische Zahlensystem

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

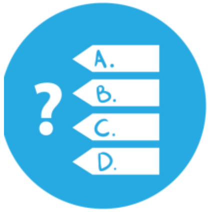
- Weiterentwicklung der Additionssysteme
- Verfügbare Zahlensymbole
- Regeln
  - I, X, C, M dürfen mehrfach nebeneinander stehen
    - I, X, C maximal dreimal
    - M beliebig oft
  - V, L, D dürfen nicht mehrfach nebeneinander stehen
  - Nebeneinanderstehende gleiche Symbole werden addiert
  - Kleinere Symbole rechts von größeren
    - Werte werden addiert
  - Kleinere Symbole links von größeren
    - Werte werden subtrahiert
    - I, X und C dürfen ihren beiden jeweils nächstgrößeren Zahlzeichen zur Substraktion vorangestellt werden
      - I vor V & X
      - X vor L & C
      - C vor D & M

Symbol	Wert
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Das römische Zahlensystem

Zahl	Wert
I	1
II	2
III	3
IV	5 – 1 = 4
V	5
VI	5 + 1 = 6
VII	5 + 2 = 7
VIII	5 + 3 = 8
IX	10 – 1 = 9
X	10

Zahl	Wert
XI	10 + 1 = 11
XII	10 + 2 = 12
XXXIX	30 + (10 – 1) = 39
XL	50 – 10 = 40
L	50
LIX	50 + (10 – 1) = 59
LX	60
XC	100 - 10 = 90
DCCC	500 + 300 = 800



- Umwandlung in das römische Zahlensystem
  - Wandeln Sie 2023 um
  - Wandeln Sie 1999 um

Stellenwertsysteme - Dezimalsystem

- Ziffer und deren absolute Position in einer Ziffernfolge entscheidend für den dargestellten Zahlenwert
- Beispiel, die Zahl **156309** im Dezimalsystem **(Basis b=10, $a_i \in \{0,1,2,\ldots9\}$)**

…	Hundert-tausender	Zehn-tausender	Tausender	Hunderter	Zehner	Einer
	10^5	10^4	10^3	10^2	10^1	10^0
	1	**5**	**6**	**3**	**0**	**9**

- Anders ausgedrückt:
  - $\mathbf{1} \cdot 10^5 + \mathbf{5} \cdot 10^4 + \mathbf{6} \cdot 10^3 + \mathbf{3} \cdot 10^2 + \mathbf{0} \cdot 10^1 + \mathbf{9} \cdot 10^0 = 156309$
  - $\mathbf{1} \cdot 100000 + \mathbf{5} \cdot 10000 + \mathbf{6} \cdot 1000 + \mathbf{3} \cdot 100 + \mathbf{0} \cdot 10 + \mathbf{9} \cdot 1 = 156309$
  - $Z = a_5 \cdot b^5 + a_4 \cdot b^4 + a_3 \cdot b^3 + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot 10^0$

$$Z = \sum_{i=0}^{n-1} a_i \cdot b^i = a_0 \cdot b^0 + a_1 \cdot b^1 + a_2 \cdot b^2 + \cdots + a_{n-1} \cdot b^{n-1}$$

Stellenwertsysteme
b-adische Darstellung

- Die Basis 10 hat sich für das Rechnen als sehr nützlich erwiesen.
  - nicht zuletzt weil wir zehn Finger haben
- Wir können aber eine beliebige andere natürlich Zahl b > 1 als Basis wählen und Zahlen darstellen als

$$Z = \sum_{i=0}^{n-1} a_i \cdot b^i \qquad a_i \in \{0,1, \ldots, b-1\}$$

- In der Informatik werden häufig folgende Basen verwendet:
  - 2 (Dualzahlen, Binärzahlen), $a_i \in \{0,1\}$
  - 8 (Oktalzahlen), $a_i \in \{0,1,2,\ldots 7\}$
  - 16 (Hexadezimalzahlen), $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
    - neben den Ziffern 0 bis 9 werden üblicherweise die Buchstaben A bis F verwendet

Dualzahlen (Binärzahlen)

- Der Basis b=2 mit einem Alphabet von zwei Zahlensymbolen, 0 und 1, kommt in der Informatik eine besondere Bedeutung zu, da sich zwei Symbole leicht elektrisch kodieren lassen:
  - "kein Strom fließt" entspricht typischerweise der 0
  - "Strom fließt" der 1
- Jede Stelle einer Binärzahl wird als "Bit" ("Binary Digit") bezeichnet
- 1 Bit ist demnach die kleinste Informationsmenge, die gespeichert werden kann
  - 8 Bits werden als 1 Byte bezeichnet

Dualzahlen (Binärzahlen)

- Beispiel, die Zahl **$(01001101)_2$** im Dualsystem (Basis b=2, $a_i \in \{0,1,\}$)

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
0	**1**	**0**	**0**	**1**	**1**	**0**	**1**

- Die Bitfolge $(01001101)_2$ entspricht der Dezimalzahl $(77)_{10}$

$(01001101)_2$
$= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
$= 0 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$
$= 64 + 8 + 4 + 1$
$= 77$

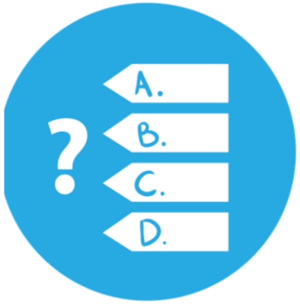
$$Z = \sum_{i=0}^{n-1} a_i \cdot b^i$$

Zweierpotenzen

2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

- Es ist für Informatiker hilfreich, mindestens die ersten 10 Zweierpotenzen auswendig zu wissen!

Zahlenkonvertierung



- Binär nach dezimal
  - Wandeln Sie 10011_2 ins Dezimalsystem um

- Dezimal nach binär
  - Wandeln Sie 47_{10} ins Binärsystem um

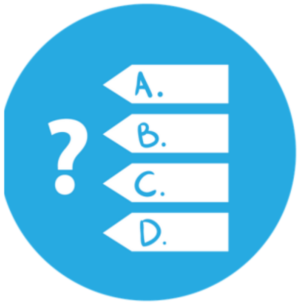
Oktal- und Hexadezimalzahlen

- Zur kompakteren Darstellung von Binärzahlen dienen oft
  - Oktalzahlen (b=8)
  - Hexadezimalzahlen (b=16)
- Die Oktalzahl $(115)_8$ entspricht der Dezimalzahl $(77)_{10}$
  $(115)_8$
  $= 1\cdot8^2 + 1\cdot8^1 + 5\cdot8^0$
  $= 1\cdot64 + 1\cdot8 + 5\cdot1$
  $= 77$
- Die Hexadezimalzahl $(4D)_{16}$ entspricht der Dezimalzahl $(77)_{10}$
  $(4D)_{16}$
  $= 4\cdot16^1 + D\cdot16^0$
  $= 64 + 13$
  $= 77$
- Welcher Zusammenhang besteht zwischen Oktal- und Hexadezimalzahlen und Binärzahlen?

Oktal- und Hexadezimalzahlen

Dezimal	Hexadezimal	Oktal	Binär
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Oktal- und Hexadezimalzahlen



- Hexadezimal nach Dezimal
  - Wandeln Sie $4AF_{16}$ in eine Dezimalzahl um

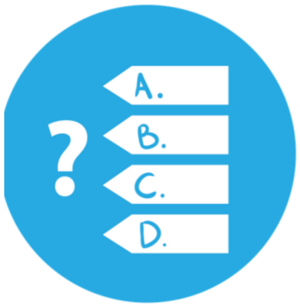
- Oktal nach Dezimal
  - Wandeln Sie 2257_8 in eine Dezimalzahl um

Rationale Zahlen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Auch rationale Zahlen können in b-adischer Darstellung angegeben werden
  - Dazu werden die Nachkommastellen mit negativen Exponenten der Basis multipliziert
- Beispiel dezimal:
  - $913{,}\mathbf{64} = 9 \cdot 10^2 + 1 \cdot 10^1 + 3 \cdot 10^0 + \mathbf{6 \cdot 10^{-1} + 4 \cdot 10^{-2}}$
- Beispiel binär:
  - $(\mathbf{11{,}101})_2$
    $= \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0 + \mathbf{1} \cdot 2^{-1} + \mathbf{0} \cdot 2^{-2} + \mathbf{1} \cdot 2^{-3}$
    $= 2 + 1 + 0.5 + 0{,}125$
    $= (3{,}625)_{10}$

Dezimalsystem



Wie wird die rationale Zahl 26,73 im Dezimalsystem b-adisch dargestellt?

Dezimal-Präfixe

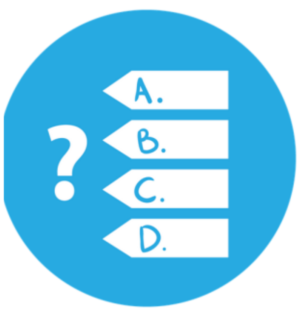
- Für viele Zehnerpotenzen gibt es standardisierte Präfixe
  - die vor physikalischen Einheiten stehen
  - z.B. 1000 m = 1 km

10^{1}	10^{2}	10^{3}	10^{6}	10^{9}	10^{12}	10^{15}	10^{18}	10^{21}	10^{24}
Deka	Hekto	Kilo	Mega	Giga	Tera	Peta	Exa	Zetta	Yotta
da	h	k	M	G	T	P	E	Z	Y

10^{-1}	10^{-2}	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}	10^{-21}	10^{-24}
Dezi	Zenti	Milli	Mikro	Nano	Piko	Femto	Atto	Zepto	Yokto
d	c	m	µ	n	p	f	a	z	y

Zweierpotenzen und Präfixe

Zweierpotenz	Präfix	Wert
2^{10}	1 Kilo (K)	1.024 ≈ 1.000
2^{20}	1 Mega (M)	1.048.576 ≈ 1.000.000
2^{30}	1 Giga (G)	1.073.741.824 ≈ 1.000.000.000



Was ist der (ungefähre) Wert von 2^{24}?

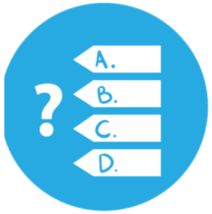
Wie viele Werte kann eine 32-Bit-Zahl annehmen?

Umwandlung zwischen Zahlensystemen

- Horner-Schema
  - Mit Hilfe dieser Darstellung
    - Lässt sich eine natürliche Zahlen n mit beliebiger Basis b …
    - … **ins Dezimalsystem umwandeln**

- Eine natürliche Zahl **n** zur Basis **b** lässt sich wie folgt darstellen:
  - $\mathbf{n} = (\ldots (((a_n * b + a_{n-1}) * b + a_{n-2}) * b + a_{n-3}) * b + \ldots + a_1) * b + a_0$

- Beispiele
  - $(1578)_{10}$ =
  - $(754)_8$ =



$(2234)_5 = ?$

Umwandlung zwischen Zahlensystemen

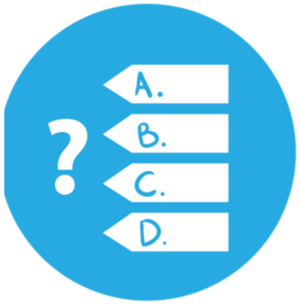
- Für die Umwandlung einer **Dezimalzahl x**
  - In ein Zahlensystem mit Basis **b**
  - Kann folgender Algorithmus verwendet werden:
    - (1) **x** / **b** = y Rest z
    - (2) Mache y zum neuen x
    - (3) Wenn x ungleich 0 gehe zu (1)
    - (4) Die ermittelten **Reste** ergeben in **umgekehrter** Reihenfolge die Zahl zur Basis b
- Beispiel
  - Umrechnung der Dezimalzahl 23521 ins Hexadezimalsystem (Basis **b= 16**)
  - `23521 / ` **`16`** ` = 1470 Rest  1 (1)` `(Rest 1 da 16*1470 = 2352`**`0)`**
  - `1470 / ` **`16`** ` = 91   Rest 14 (E)` `(Rest 14, da 16*91 = 1456)`
  - `91 / ` **`16`** ` = 5    Rest 11 (B)`
  - `5 / ` **`16`** ` = 0    Rest  5 (5)`
  - Ergebnis: $(5BE1)_{16}$

Umwandlung zwischen Zahlensystemen

- Beispiel
  - Umrechnung der Dezimalzahl 6485 ins **Binärsystem**
  - `6485 / `**2**` = 3242 Rest 1`
  - `3242 / `**2**` = 1621 Rest 0`
  - `1621 / `**2**` =  810 Rest 1`
  - ` 810 / `**2**` =  405 Rest 0`
  - ` 405 / `**2**` =  202 Rest 1`
  - ` 202 / `**2**` =  101 Rest 0`
  - ` 101 / `**2**` =   50 Rest 1`
  - `  50 / `**2**` =   25 Rest 0`
  - `  25 / `**2**` =   12 Rest 1`
  - `  12 / `**2**` =    6 Rest 0`
  - `   6 / `**2**` =    3 Rest 0`
  - `   3 / `**2**` =    1 Rest 1`
  - `   1 / `**2**` =    0 Rest 1`
  - Ergebnis: $(1100101010101)_2$

Umwandlung zwischen Zahlensystemen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES



- Hexadezimal nach Binär
  - Wandeln Sie $4AF_{16}$ in eine Dualzahl um

- Binär nach Oktal
  - Wandeln Sie $0100\ 1010\ 1111_2$ in eine Oktalzahl um

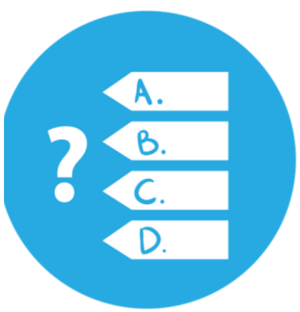
Umwandlung rationaler Zahlen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Zur Umwandlung rationaler Zahlen wird
  - Der Anteil vor dem Komma und
  - Der Anteil nach dem Komma **separat** betrachtet
- Die Umwandlung des Vorkommaanteils erfolgt mit dem bekannten Algorithmus
- Für die Umwandlung des Nachkommateils **x** einer Dezimalzahl
  - in ein anderes Stellenwertsystem existiert folgender Algorithmus,
  - wobei **b** die Basis des Zielsystems ist
    - (1) x * b = y Überlauf z (eventueller ganzzahliger Anteil nach Multiplikation)
    - (2) Mache den Nachkommaanteil y zum neuen x
    - (3) Wenn x ungleich 0 und nicht genügend Nachkommastellen ermittelt, gehe zu (1)
    - (4) Die ermittelten Überläufe ergeben in der Berechnungsreihenfolge die Zahl zur Basis b

Umwandlung rationaler Zahlen

- Beispiel
  - Umrechnung der Dezimalzahl 0,6875 ins Binärsystem
  - `0,6875 * 2 = 1,375`
  - `0,375  * 2 = 0,75`
  - `0,75   * 2 = 1,5`
  - `0,5    * 2 = 1,0`
  - Das Ergebnis (von oben nach unten) ist $(0{,}1011)_2$



Umrechnung der Dezimalzahl 0,375 ins Binärsystem und zurück

Umwandlung rationaler Zahlen

- Beispiel:
  - Umrechnung der Dezimalzahl 0,1 ins Binärsystem
  - `0,1 * 2 = 0,2`
  - `0,2 * 2 = 0,4`
  - `0,4 * 2 = 0,8`
  - `0,8 * 2 = 1,6`
  - `0,6 * 2 = 1,2`
  - `0,2 * 2 = 0,4`
  - `0,4 * 2 = ...`
  - Der Algorithmus endet nie
    - Der Nachkommaanteil ist im Binärsystem periodisch
    - Beim Runden auf eine endliche Zahl von Nachkommastellen kommt es zu einem Rundungsfehler

Gliederung

- Zahlen
  - Einfache Zahlendarstellungen
  - Stellenwertsysteme
  - Rationale Zahlen
  - Umwandlungen zwischen Zahlensystemen
- **Computerzahlen**
  - **Vorzeichenlose, ganze Zahlen**
  - **Vorzeichenbehaftete ganze Zahlen**
  - **Vorzeichenbehaftete rationale Zahlen**
- Zeichen
  - ASCII-Code
  - ISO 8859
  - Unicode

Computerzahlen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Für die Darstellung von Zahlen im Computer wird üblicherweise eine feste Anzahl von Bits verwendet
- → eingeschränkter Wertebereich
- → eingeschränkte Genauigkeit

- Wir betrachten nachfolgend
  - Vorzeichenlose, ganze Zahlen
  - Vorzeichenbehaftete ganze Zahlen
  - Vorzeichenbehaftete rationale Zahle

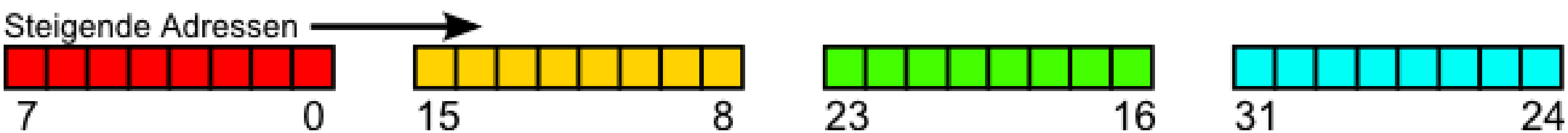
Vorzeichenlose ganze Zahlen

- Nahezu alle gängigen Computerarchitekturen fassen je 8 Bit zu einem Byte zusammen
- In Abhängigkeit von der Anzahl der Bits können unterschiedliche Zahlenbereiche dargestellt werden

Bytes	Bits	Wertebereich	C#-Type
1	8	[0; 255]	byte
2	16	[0; 65536]	ushort
4	32	[0; 4294967295]	uint
8	64	[0; 18446744073709551615]	ulong

Big-Endian- vs. Little-Endian-Speicherung

- Die Speicherordnung einer Computerarchitektur legt fest, in welcher Reihenfolge Bytes im Arbeitsspeicher abgelegt werden
  - (engl „Byte order“ oder „*endianess*“)
- Little-Endian
  - Das Byte mit der niedrigsten Wertigkeit wird **zuerst** gespeichert
  - Alle x86-kompatiblen Rechner verwenden diese Ordnung
    - Von Intel maßgeblich entwickelt / geprägt



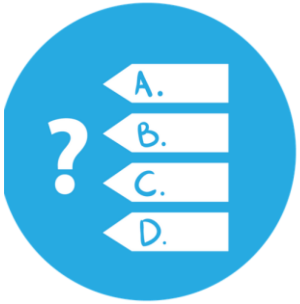
- Big-Endian
  - Das Byte mit der niedrigsten Wertigkeit wird **zuletzt** gespeichert
  - MIPS, SPARC, PowerPC, Java Virtual Machine, etc.

Addition von Dualzahlen

- Für die duale Addition gilt allgemein
  - `0 + 0                    = 0`
  - `0 + 1                    = 1`
  - `1 + 0                    = 1`
  - `1 + 1                    = 0 Übertrag 1`
  - `1 + 1 + 1 (vom Übertrag) = 1 Übertrag 1`

- Beispiel
  - `0 1 0 1 1 0 1 = 45`
  - `0 1 1 0 1 1 0 = 54`
  - `------------------------`
  - `1 1 1 1       = Übertrag`
  - `------------------------`
  - `1 1 0 0 0 1 1 = 99`

Addition von Dualzahlen



- Addieren Sie 11011 mit 10001

- Addieren Sie 10101 mit 11100

Addition von Dualzahlen
Überlauf

- Beispiel
  - Addition von zwei 4-bit-Dualzahlen
  - `  1 0 1 1 = 11`
  - `  0 1 1 0 =  6`
  - `--------------`
  - `1 0 0 0 1`
  - **Überlauf**
- Eine Addition läuft über
  - Wenn ihr Ergebnis nicht mit der Anzahl der verfügbaren Bits gespeichert werden kann
  - Zahlenbereich einer vorzeichenlosen 4-Bit-Dualzahl?

Vorzeichenbehaftete ganze Zahlen

- Für die Darstellung positiver und negativer Zahlen gibt es verschiedene Möglichkeiten
  - Betrags-Vorzeichendarstellung
  - Einerkomplement
  - Zweierkomplement

- Jede dieser Darstellungen hat Vor- und Nachteile bezüglich:
  - der **Symmetrie** des darstellbaren Wertebereichs
    - Jede Zahl z kann ebenfalls die Zahl –z darstellen
  - der **Eineindeutigkeit** der Darstellung
    - Jede Zahl entspricht genau einem Bitmuster und umgekehrt
  - der Durchführung von **arithmetischen** Operationen
    - Kann (einfach) gerechnet werden?

Betrags-Vorzeichendarstellung

- Das höchstwertige Bit wird für das Vorzeichen verwendet
  - 0 <-> positiv
  - 1 <-> negativ
- Die verbleibenden Bits werden für den Betrag verwendet
- Symmetrischer Wertebereich $[-(2^{n-1}-1); +(2^{n-1}-1)]$
  - jede Zahl +z kann auch negiert –z dargestellt werden
  - Vorzeichenbit muss lediglich gekippt werden (Negation)
- Beispiel für n=4

Dezimal	0	1	2	3	4	5	6	7
Binär	0000	0001	0010	0011	0100	0101	0110	0111

Dezimal	-0	-1	-2	-3	-4	-5	-6	-7
Binär	1000	1001	1010	1011	1100	1101	1110	1111

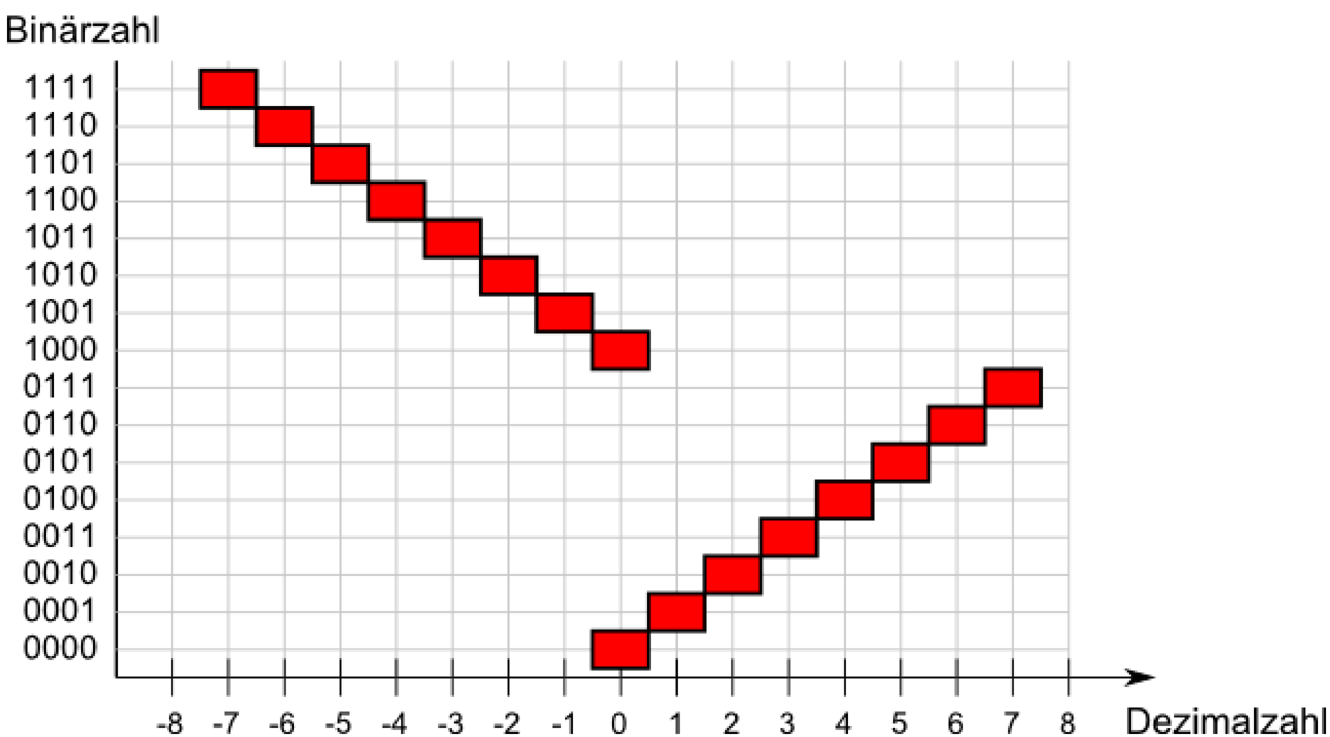
Betrags-Vorzeichendarstellung

- „normale“ Addition/Subtraktion funktioniert nicht mit Vorzeichenbit
  - Beispiel 5 + (-6) = -1 mittels Addition mit Dualzahlen

```
   0 1 0 1 (+5)
   1 1 1 0 (-6)
 ---------
 1 0 0 1 1 (-3) falsch!
```

Betrags-Vorzeichendarstellung

- Darstellung **nicht** eineindeutig
  - Null ist doppelt repräsentiert
    - + 0 = **0**000
    - - 0 = **1**000
  - Problem bei Test auf Gleichheit => - 0 ≠ 0



Einerkomplement

- Eine negative Zahl wird durch das bitweise Komplement der positiven Zahl dargestellt
- Symmetrischer Wertebereich: $[-(2^{n-1}-1); +(2^{n-1}-1)]$
- Beispiel für n=4

Dezimal	0	1	2	3	4	5	6	7
Binär	0000	0001	0010	0011	0100	0101	0110	0111

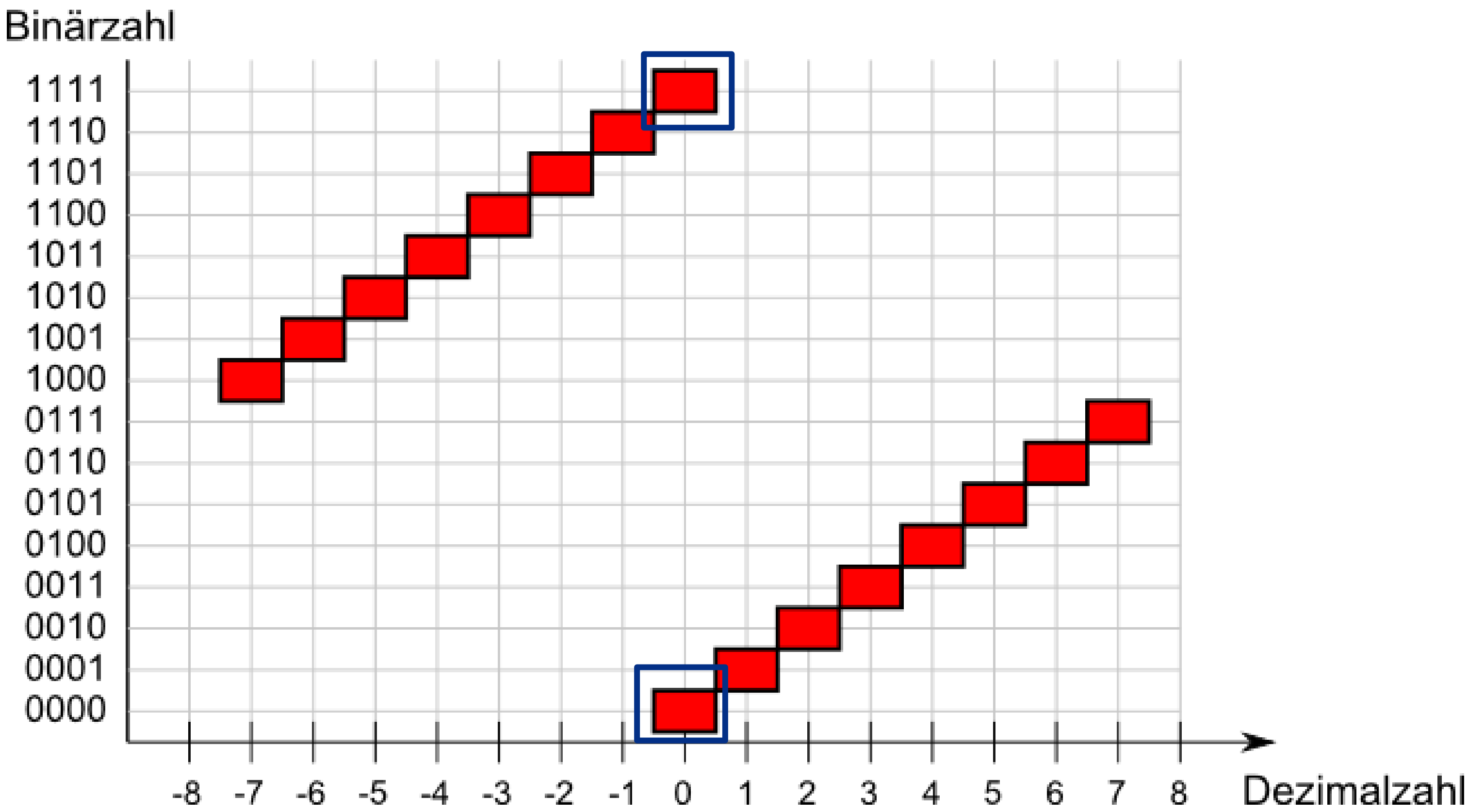
Dezimal	-0	-1	-2	-3	-4	-5	-6	-7
Binär	1111	1110	1101	1100	1011	1010	1001	1000

Einerkomplement

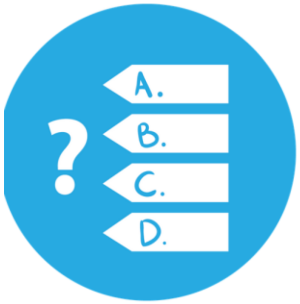
- Darstellung nicht eindeutig
  - Null ist doppelt repräsentiert
  - Problem bei Test auf Gleichheit
- Addition/Subtraktion muss in zwei Schritten erfolgen
  - Normale Binäraddition
  - Falls Überlauf wird dieser addiert; der Übertrag wird gestrichen
- Beispiel
  - `  0 1 0 1    5`
  - `  1 1 0 0   -3`
  - `--------------`
  - `1 0 0 0 1  -14`
  - `        1    1`
  - `--------------`
  - ~~1~~ `0 0 1 0    2`

Einerkomplement

- 0 ist doppelt repräsentiert
  - Beim Übergang von negativen in den positiven Bereich ist das Ergebnis um 1 zu gering => Daher die Übertragsadditionsregel



Rechenbeispiele



- Berechnen Sie im 1er Komplement $(2)_{10}$ $+(-5)_{10}$ mit 4 verfügbaren Zeichen

- Berechnen Sie im 1er Komplement $(-3)_{10}$ + $(6)_{10}$ mit 4 verfügbaren Zeichen

Zweierkomplement

- Beim Zweierkomplement werden negative Zahlen wie folgt gebildet
  - Bildung des Einerkomplements (**inkl. Negation!**)
  - Addition von 1
- Beispiel:
  - $(-6)_{10}$ = Einerkomplement (0110) + 1 = 1001 + 1 = 1010
- **Asymmetrischer** Wertebereich: $[-(2^{n-1}); +(2^{n-1}-1)]$
- Beispiel für n=4

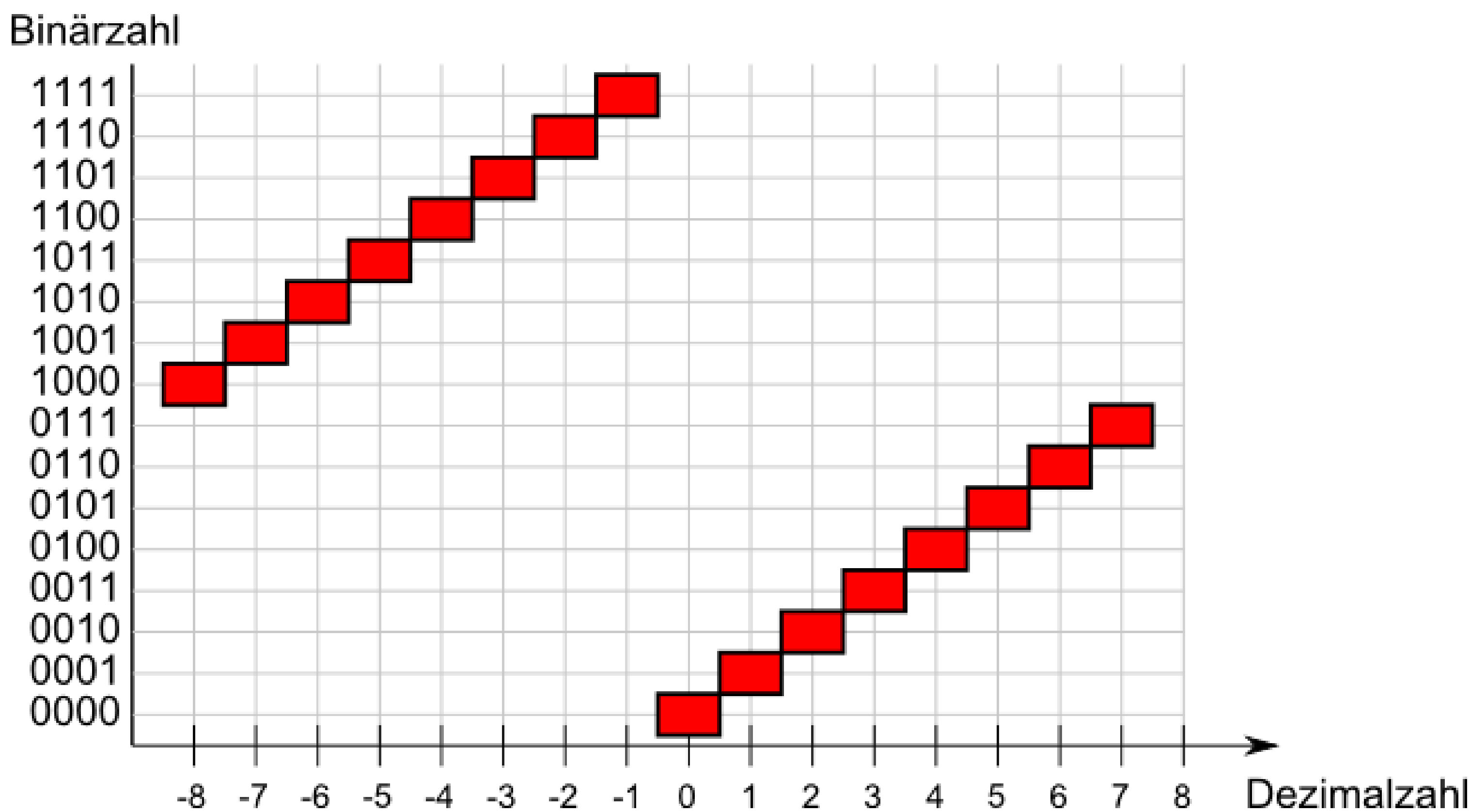
Dezimal	**0**	1	2	3	4	5	6	**7**
Binär	0000	0001	0010	0011	0100	0101	0110	0111

Dezimal	**-1**	-2	-3	-4	-5	-6	-7	**-8**
Binär	1111	1110	1101	1100	1011	1010	1001	1000

Zweierkomplement

- **Darstellung** ist **eindeutig;** Null nur einmal vorhanden
- Zur Addition/Subtraktion kann die normale vorzeichenlose Binäraddition verwendet werden
- Subtraktion entspricht Negation und anschließender Addition
- Beispiel:
  - `(5 - 6) = (5 + (-6))`
  - `  0 1 0 1     5`
  - `+ 1 0 1 0    -6`
  - `--------------`
  - `  1 1 1 1    -1`

Zweierkomplement



Zweierkomplement

- Beispiel
  - `    0 1 0 1      5`
  - `  + 1 1 0 1     -3`
  - `  --------------`
  - `  ~~1~~ 0 0 1 0      2`
- Beispiel
  - `    0 1 0 0      4`
  - `  + 1 1 0 1     -3`
  - `  --------------`
  - `  ~~1~~ 0 0 0 1      1`
- Es scheint als könnten wir (anders als beim Einerkomplement) den Übertrag beim Zweierkomplement einfach streichen
- Ist das immer so?

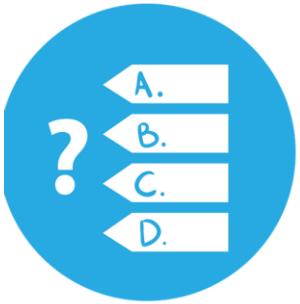
Zweierkomplement

- Beispiel
  - `    1 1 0 0    -4`
  - `  + 1 0 1 1    -5`
  - `  --------------`
  - `  ~~1~~ 0 1 1 1     7    falsch!`
- Beispiel
  - `    0 1 0 0     4`
  - `  + 0 1 0 1     5`
  - `  --------------`
  - `    1 0 0 1    -7    falsch!`
- Der Überlauf kann nur gestrichen werden, wenn das Ergebnis im darstellbaren Zahlenbereich liegt
  - Wertebereich: $[-(2^{n-1}); +(2^{n-1}-1)]$ bei n = 4
  - = [-8; + 7]

Zweierkomplement

- Inverse des Zweierkomplements
  - Die Wirkung der Zweierkomplementbildung kann durch erneute Anwendung des Zweierkomplements aufgehoben werden
- Beispiel
  - $(6)_{10}$ = 0110
  - $(-6)_{10}$ = Einerkomplement (0110) + 1 = 1001 + 1 = 1010
  - $(6)_{10}$ = Einerkomplement (1010) + 1 = 0101 +1 = 0110

Rechenbeispiele



- Berechnen Sie im 2er-Komplement $(5)_{10} + (-5)_{10}$ mit 4 verfügbaren Zeichen

- Berechnen Sie im 2er-Komplement $(45)_{10} + (-23)_{10}$ mit 8 verfügbaren Zeichen.
  - Geben Sie einen möglichst symmetrischen Wertebereich an

Vorzeichenbehaftete ganze Zahlen
Zusammenfassung

Darstellung	Wertebereich	Eindeutig	Operationen
Betrags-Vorzeichen	symmetrisch	nein	Sonderbehandlung
Einerkomplement	symmetrisch	nein	2-Stufen-Addition
Zweierkomplement	**asymmetrisch**	**ja**	**einfach**

- In der Praxis ist das Zweierkomplement die vorherrschende Darstellung

Bytes	Bits	Wertebereich	Java-Type
1	8	[–128;127]	byte
2	16	[–32768;32767]	short
4	32	[–2147483648;2147483647]	int
8	64	[–9223372036854775808; 9223372036854775807]	long

Vorzeichenbehaftete rationale Zahlen

- Für die Repräsentation von rationalen Zahlen im Binärsystem haben sich zwei Darstellungsformen etabliert:
  - Festkommazahlen
  - Gleitkommazahlen

Festkommazahlen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Fester Bereiche für Ziffern vor dem Komma und nach dem Komma
- Ähnlich wie Betrags-Vorzeichendarstellung bei ganzen Zahlen
- Das höchstwertige Bit *s* wird als Vorzeichen verwendet

15															0
s	m_{14}	m_{13}	m_{12}	m_{11}	m_{10}	m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0

- Die verbleibenden n-1 Bits werden als Mantisse verwendet
  - Speicherung der Vor- und Nachkommastellen
  - k ≤ (n-1) Vorkommastellen
  - j = (n-1) – k Nachkommastellen

$$w = (-1)^s \left(\left(\sum_{i=0}^{k-1} m_{j+i} \cdot 2^i \right) + \left(\sum_{i=-j}^{-1} m_{j+i} \cdot 2^i \right) \right) = (-1)^s \left(\sum_{i=-j}^{k-1} m_{j+i} \cdot 2^i \right)$$

Festkommazahlen

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- Beispiel: Umrechnung von 11011001 für n=8, k=4 und j=3

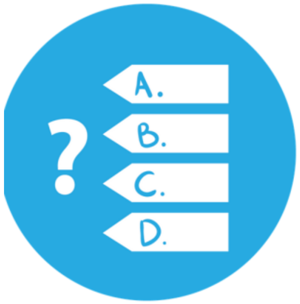
s	m_6	m_5	m_4	m_3	m_2	m_1	m_0

$$w = (-1)^s \left(\sum_{i=-j}^{k-1} m_{j+i} \cdot 2^i \right)$$

$$= (-1)^1 \left(1 \cdot 2^{-3} + 0 \cdot 2^{-2} + 0 \cdot 2^{-1} + 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 \right)$$

$$= (-1)\left(0{,}125 + 1 + 2 + 8\right) = -\,11{,}125$$

Rechenbeispiel



- Beispiel: Umrechnung von **00011011** für n=8, k=5 und j=2

Festkommazahlen

- Ermöglicht die Darstellung eines Ausschnitts der rationalen Zahlen in der IT
- Festkommadarstellung ist ein sog. äquidistantes Zahlenformat
  - Abstand zwischen zwei darstellbaren Zahlen immer gleich
- Bei ähnlichem Wertebereich der beteiligten Zahlen
  - Garantierte Genauigkeit der Darstellung
- Häufiger Einsatz bei digitalen Signalprozessoren oder Buchführungssoftware
- Nachteil: Darstellung von Zahlen nur exakt innerhalb des Ausschnitts
  - Pi oder 1/3 können nicht exakt dargestellt werden => Rundungsfehler.

Gleitkommazahlen

- Häufig müssen Zahlen mit sehr unterschiedlichen Größenordnungen verarbeitet werden
- Beispiel Naturkonstanten
  - Lichtgeschwindigkeit: $\overbrace{2{,}99792458}^{\text{Mantisse}}\cdot 10^{8}$ m/s
  - Elektronenmasse: $9{,}10938291\cdot 10^{-31}$ kg
  - Elementarladung: $1{,}60217656\cdot 10^{-19}$ C
- Bei der wissenschaftlichen Potenzschreibweise wird das Komma durch den Exponenten verschoben
  - Verschiebung nach **rechts** für **kleine** Zahlen: 0,000.41=4,1 * 10^{-4}
  - Verschiebung nach **links** für **große** Zahlen: 1.200.000,000.41≈$1{,}2\cdot 10^{6}$
- Die binäre Gleitkommadarstellung (engl. "floating point")
  - kopiert die wissenschaftlichen Potenzschreibweise und
  - erweitert die Festkommadarstellung um einen **binär-kodierten Exponenten**

Gleitkommazahlen

- Beispiel, n = 16
  - 1 Bit Vorzeichen s, 5 Bit Exponent e, 10 Bit Mantisse m

s	e_4	e_3	e_2	e_1	e_0	m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
15															0

  - Die fünf Bit des Exponenten können $2^5 = 32$ Zahlen darstellen
  - Um negative **Exponenten e** darzustellen, werden statt dem Exponenten dessen ***Charakteristik* c** >0 in den 5 Bits gespeichert
  - Damit kann der Exponent Werte aus dem Intervall [−15; 16] annehmen

s	c_4	c_3	c_2	c_1	c_0	m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0
15															0

  - Die Charakteristik berechnet sich aus c= e + k
    - Die **Konstante k** entspricht immer dem (vorzeichenlosen) Wert des kleinsten darstellbaren Exponenten (siehe Intervall)

Gleitkommazahlen

Beispiel z= 0,001101 als Gleitkommazahl

- z = 0,001101
  = **00000,1101 * 2^{-2}** (Komma um zwei nach rechts verschoben)

  s = 0 (positiv)

  c = e + k = -2 + 15 = 13 = 01101

  m = 1101

0	0	1	1	0	1	1	1	0	1	0	0	0	0	0	0

- z = 0,001101
  = **0000,01101* 2^{-1}** (Komma um eins nach rechts verschoben)

  c = e + k = -1 + 15 = 14 = 01110

  m = 01101

0	0	1	1	1	0	0	1	1	0	1	0	0	0	0	0

Gleitkommazahlen

Beispiel z= 0,001101 als Gleitkommazahl

- z = 0,001101
  = **00,0001101 * 2^1** (Komma um eins nach **links** verschoben)

  s = 0 (positiv)
  c = e + k = 1 + 15= 16 = 10000
  m = 0001101

0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0

- z = 0,001101
  = **0,00001101* 2^2** (Komma um zwei nach links verschoben)
  c = e + k = 2 + 15= 17 = 10001
  m = 00001101

0	1	0	0	0	1	0	0	0	0	1	1	0	1	0	0

- **Feststellung?**:
  - Die selbe Zahl (Ziffernfolge) wird immer anders abgespeichert!
  - Problem: Die Vergleichbarkeit identischer Zahlen ist nicht gegeben!

Gleitkommazahlen

- Lösungsansatz: Festlegung von Normalisierungsregeln
  - Die Position des Kommas wird mit einer Normalisierungsregel **eindeutig** festgelegt

- Normalisierungsregeln
  - Nachkommanormalisierung:
    - Exponent wird so gewählt, dass die **erste Nachkommastelle** ungleich 0 ist, hier also bei 0,001101 => **00000,<u>1</u>101 * 2**$^{-2}$
  - Vorkommanormalisierung
    - Exponent wird so gewählt, dass die **erste Vorkommastelle** ungleich 0 ist, hier also 0,001101 => **00000<u>1</u>,101 * 2**$^{-3}$
  - Vorteil?

Gleitkommazahlen

- Beispiel 1: Nachkomma-Normalisierung ungepackt
  - z = 0,001011
    = $0{,}1011 * 2^{-2}$ (Komma um zwei nach rechts verschoben)
    s = 0
    c = -2 + 15 = 13 = 01101
    m = <u>1011</u>

0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0

- Beispiel 2: Vorkomma-Normalisierung, ungepackt
  - z = 0,001011
    = $1{,}011 * 2^{-3}$ (Komma um drei nach rechts verschoben)
    s = 0
    c = – 3 + 15 = 12 = 01100
    m = <u>1011</u>

0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0

Gleitkommazahlen

- Beispiel 3: Nachkomma-Normalisierung, gepackt
  - z = 0,001011
    = 0,1011 * 2^{-2} (Komma um zwei nach rechts verschoben)
    s = 0
    c = 15 – 2 = 13 = 01101
    m = (1)011

0	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0

- Beispiel 4: Vorkomma-Normalisierung, gepackt
  - z = 0,001011
    = 1,011 * 2^{-3} (Komma um drei nach rechts verschoben)
    s = 0
    c = 15 – 3 = 12 = 01100
    m = (1)011

0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0

Gleitkommazahlen

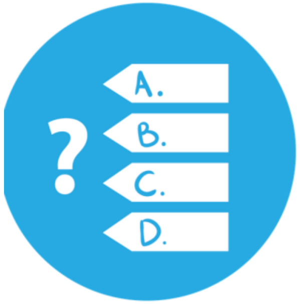
- Vorkomma-Normalisierung, gepackt, reverse

0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0

  - s = 0
    $e = c - k = (01100)_2 - (15)_{10} = (12)_{10} - (15)_{10} = -3$
    m = (**1**),1010000000 = **1**,1010000000
    $m = 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$
    $m = 1 + 0{,}5 + 0{,}125 = 1{,}625 * 2^{-3}$

- Vor- und Nachteile der (gepackten) Darstellung?

Rechenbeispiel

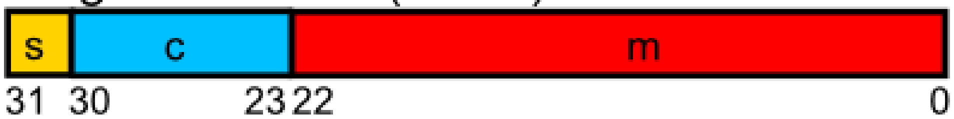


- Vorkomma-Normalisierung, gepackt, reverse (n=16, 1 Bit Vorzeichen s, 5 Bit Charakteristik k, 10 Bit Mantisse m).

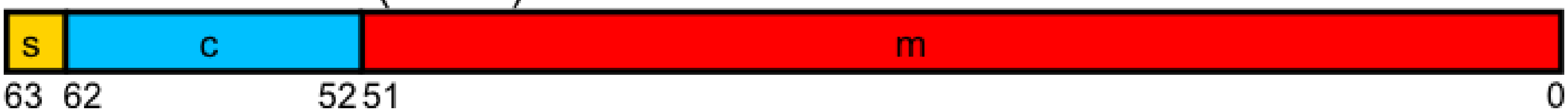
0	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0

Gleitkommazahlen
IEEE Std. 754

Single Precision (32-bit)



Double Precision (64-bit)



Charakteristik c	Mantisse m	32-/64-bit Precision
000 ... 0000	beliebig	32: $(-1)^s\ 0,m * 2^{-126}$ 64: $(-1)^s\ 0,m * 2^{-1022}$
111 ... 1111	$= 0$	$(-1)^s * \infty$
111 ... 1111	$\neq 0$	Not a Number (NaN)
alle anderen	beliebig	32: $(-1)^s\ 1,m * 2^{c-127}$ 64: $(-1)^s\ 1,m * 2^{c-1023}$

Gleitkommazahlen
IEEE Std. 754

- Die IEEE 754 Formate (aus dem Jahre 1985) werde von allen gängigen Mikroprozessoren unterstützt
- Normalerweise gepackte Darstellung mit Vorkomma-Normalisierung
- Es gibt einige reservierte Bitmuster für die Charakteristik mit Sonderbedeutung:
  - Null
    - Charakteristik `c = 000…0000` schaltet in eine ungepackte Darstellung um
    - Darstellung der Null durch `m = 0`
  - Unendlich
    - Charakteristik `c = 1111…111` und `m = 0`
    - Vorzeichen s entscheidet zwischen $+\infty$ und $-\infty$
  - Not a Number (NaN)
    - Charakteristik `c = 1111…111` und $m \neq 0$
    - Entsteht bei algebraischen Operationen mit undefiniertem oder nicht repräsentierbarem Ergebnis, wie z.B. 0/0, $0 \cdot \infty$, usw.

Gleitkommazahlen
IEEE Std. 754

- Für die Darstellung von Gleitpunktzahlen in einem (Quell-)Text wird typischerweise die Notation "52.21e-2" oder "52.21E-2" verwendet
  - Die Zahl hinter dem "E" gibt den Exponenten der Zehnerpotenz an
- Diese Notation ist auch in vielen Programmiersprachen üblich
- Anstatt des deutschen Kommas wird im Englischen (und daher in fast allen Programmiersprachen) der Dezimalpunkt verwendet
- Größenordnungen

Bytes	Bits	Wertebereich	Type
4	32	± 1.4e-45 ... 3.403e38	float
8	64	± 4.94e-324 ... 1.798e308	double

Gliederung

- Zahlen
  - Einfache Zahlendarstellungen
  - Stellenwertsysteme
  - Rationale Zahlen
  - Umwandlungen zwischen Zahlensystemen
- Computerzahlen
  - Vorzeichenlose, ganze Zahlen
  - Vorzeichenbehaftete ganze Zahlen
  - Vorzeichenbehaftete rationale Zahlen
- **Zeichen**
  - **ASCII-Code**
  - **ISO 8859**
  - **Unicode**

Codierung von Zeichen

- Um Text auf einem Computer darzustellen, muss jeder Buchstabe binär kodiert werden
- Je nachdem wie viele Bits pro Zeichen verwendet werden, können unterschiedlich viele verschiedene Zeichen abgelegt werden
- Beispiele:
  - 7 Bits: $2^7 = 128$ verschiedene Zeichen
    - ASCII
  - 8 Bits: $2^8 = 256$ verschiedene Zeichen
    - ISO 8859
  - 16 Bits: $2^{16} = 65536$ verschiedene Zeichen
    - Unicode UTF-16

ASCII

- American Standard Code for Information Interchange
  - 1963 von der American Standards Association (ASA) beschlossen
- 7-Bit-Zeichencodierung
  - Aber jedes Zeichen wird als ein Byte gespeichert
  - Höchstwertiges Bit ist immer 0
- Insgesamt 128 Zeichen
  - 95 druckbare
  - 33 nicht druckbare

ASCII

- Steuerzeichen stammen aus der Zeit der Fernschreiber
  - Viele werden heute nicht mehr benötigt
- Wichtig:
  - Zeilenvorschub "LF" (Line Feed, ASCII $(0A)_{16}$
  - Wagenrücklauf “CR“ (Carriage Return, ASCII $(0D)_{16}$
- Der Buchstabe A gemäß ASCII-Tabelle
  - (0) 100 0001

ASCII-Tabelle

b7 b6 b5 (BITS) / b4 b3 b2 b1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	Steuerzeichen		Symbole		Grossbuchstaben		Kleinbuchstaben	
0 0 0 0	0 NUL 0 0	16 DLE 10 20	32 SP 20 40	48 0 30 60	64 @ 40 100	80 P 50 120	96 ‘ 60 140	112 p 70 160
0 0 0 1	1 SOH 1 1	17 DC1 11 21	33 ! 21 41	49 1 31 61	65 A 41 101	81 Q 51 121	97 a 61 141	113 q 71 161
0 0 1 0	2 STX 2 2	18 DC2 12 22	34 ” 22 42	50 2 32 62	66 B 42 102	82 R 52 122	98 b 62 142	114 r 72 162
0 0 1 1	3 ETX 3 3	19 DC3 13 23	35 # 23 43	51 3 33 63	67 C 43 103	83 S 53 123	99 c 63 143	115 s 73 163
0 1 0 0	4 EOT 4 4	20 DC4 14 24	36 $ 24 44	52 4 34 64	68 D 44 104	84 T 54 124	100 d 64 144	116 t 74 164
0 1 0 1	5 ENQ 5 5	21 NAK 15 25	37 % 25 45	53 5 35 65	69 E 45 105	85 U 55 125	101 e 65 145	117 u 75 165
0 1 1 0	6 ACK 6 6	22 SYN 16 26	38 & 26 46	54 6 36 66	70 F 46 106	86 V 56 126	102 f 66 146	118 v 76 166
0 1 1 1	7 BEL 7 7	23 ETB 17 27	39 ’ 27 47	55 7 37 67	71 G 47 107	87 W 57 127	103 g 67 147	119 w 77 167
1 0 0 0	8 BS 8 10	24 CAN 18 30	40 (28 50	56 8 38 70	72 H 48 110	88 X 58 130	104 h 68 150	120 x 78 170
1 0 0 1	9 HT 9 11	25 EM 19 31	41) 29 51	57 9 39 71	73 I 49 111	89 Y 59 131	105 i 69 151	121 y 79 171
1 0 1 0	10 LF A 12	26 SUB 1A 32	42 * 2A 52	58 : 3A 72	74 J 4A 112	90 Z 5A 132	106 j 6A 152	122 z 7A 172
1 0 1 1	11 VT B 13	27 ESC 1B 33	43 + 2B 53	59 ; 3B 73	75 K 4B 113	91 [5B 133	107 k 6B 153	123 { 7B 173
1 1 0 0	12 FF C 14	28 FS 1C 34	44 , 2C 54	60 < 3C 74	76 L 4C 114	92 \ 5C 134	108 l 6C 154	124 | 7C 174
1 1 0 1	13 CR D 15	29 GS 1D 35	45 – 2D 55	61 = 3D 75	77 M 4D 115	93] 5D 135	109 m 6D 155	125 } 7D 175
1 1 1 0	14 SO E 16	30 RS 1E 36	46 . 2E 56	62 > 3E 76	78 N 4E 116	94 ^ 5E 136	110 n 6E 156	126 ~ 7E 176
1 1 1 1	15 SI F 17	31 US 1F 37	47 / 2F 57	63 ? 3F 77	79 O 4F 117	95 _ 5F 137	111 o 6F 157	127 DEL 7F 177

Legende: dez Zeichen hex okt

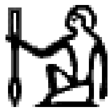
ISO 8859

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

- ASCII nutzt nur 7 der 8 Bits eines Byte
- Der restliche Zahlenbereich kann für andere Zeichencodierungen verwendet werden
- Die International Organisation for Standardization (ISO) definiert 15 solche Erweiterungen
- ISO 8859-1 enthält z.B. die für uns in Deutschland wichtigen Buchstaben: "ä", "ö", "ü", "ß"

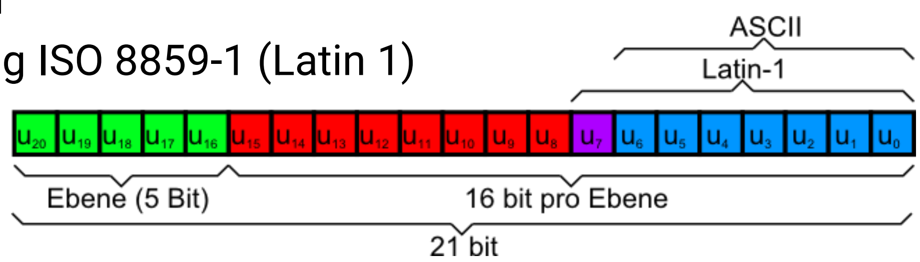
ISO 8859-1	Westeuropäisch (Latin-1)
ISO 8859-2	Mitteleuropäisch (Latin-2)
ISO 8859-3	Südeuropäisch (Latin-3)
ISO 8859-4	Nordeuropäisch (Latin-4)
ISO 8859-5	Kyrillisch
ISO 8859-6	Arabisch
ISO 8859-7	Griechisch
ISO 8859-8	Hebräisch
ISO 8859-9	Türkisch (Latin-5)
ISO 8859-10	Nordisch (Latin-6)
ISO 8859-11	Thai
ISO 8859-12	verworfen
ISO 8859-13	Baltisch (Latin-7)
ISO 8859-14	Keltisch (Latin-8)
ISO 8859-15	Westeuropäisch (Latin-9)
ISO 8859-16	Südosteuropäisch (Latin-10)

Unicode

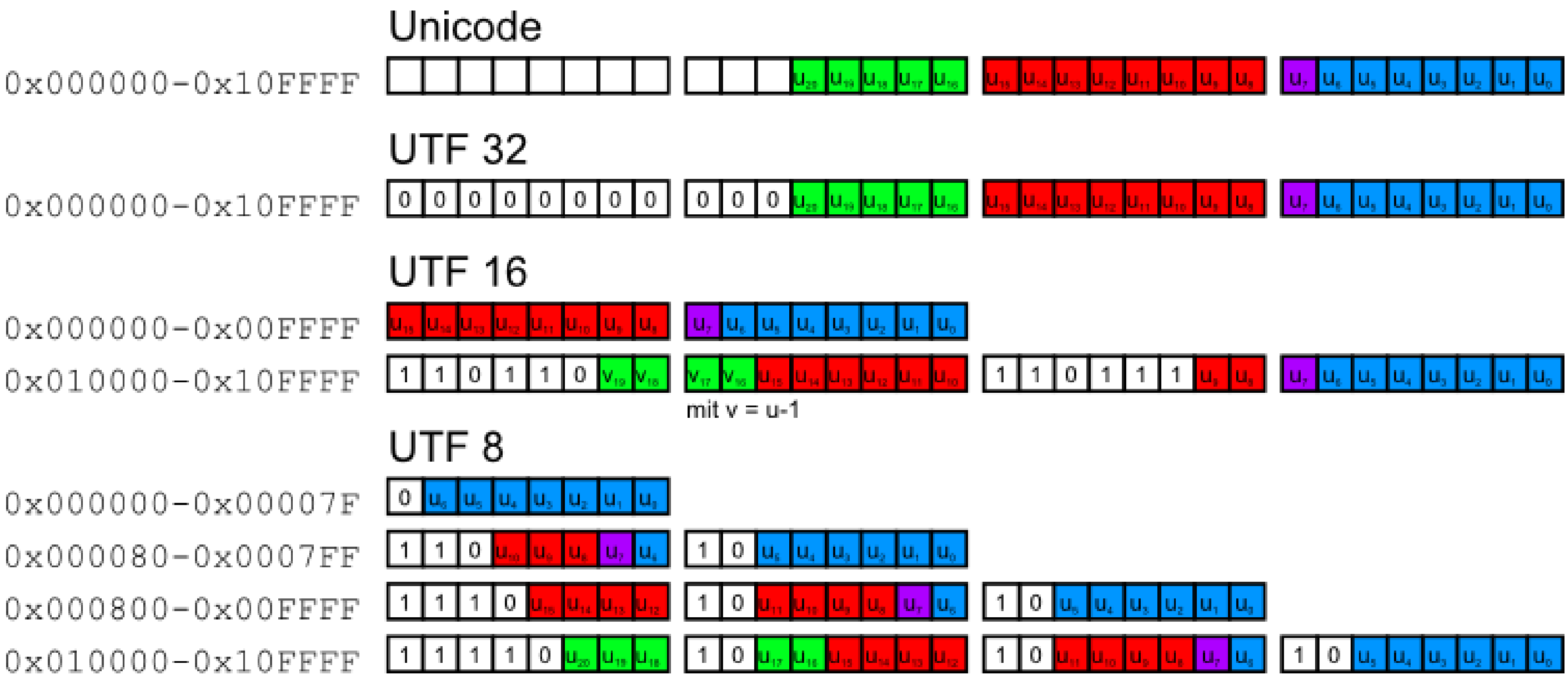
- Die Verwendung von ISO 8859 führt immer wieder zu Problemen
  - Wenn Sender und Empfänger nicht die gleiche ISO 8859-x verwenden
- Außerdem sind bei weitem nicht alle Schriftzeichen erfasst
- Idee
  - Eine einzige universelle Codierung für alle relevanten Zeichen
- Unicode, ISO 10646
  - Besteht aus 17 Ebenen „planes" (darstellbar durch 5 Bit)
  - Jede Ebene wird durch 16 Bit codiert
    - 2^{16} = 65536 Zeichen
  - Ein Unicode benötigt also 5 + 16 = 21 Bit
  - Die meisten aktuell verwendeten Zeichen befinden sich in Ebene 0
    - Basic Multilingual Plane (BMP)
  - Unicode-Zeichen werden üblicherweise durch „U+" und eine mindestens vierstellige Hexadezimalzahl angegeben
  - U+1300C für die ägyptische Hieroglyphe 

Unicode

- Die Kodierung aller möglichen Schriftzeichen ist ein andauernder Prozess
  - d.h. die Anzahl der Zeichen wächst ständig
- Ein Problem bei der Darstellung ist, dass die meisten Schriftarten nur eine kleine Untermenge der im Unicode definierten Zeichen bereit halten
  - Ist ein Zeichen in einer Schrift nicht vorhanden, wird oftmals einfach ein Zeichen aus einer anderen Schriftart eingefügt
- Die Webseite http://www.decodeunicode.org/ hat es sich zur Aufgabe gemacht, alle aktuell im Unicode kodierten Zeichen darzustellen
- Beim Entwurf des Unicode wurde auf Kontinuität wert gelegt
  - Aus den 21 Bits des Unicodes entsprechen
    - die ersten 7 Bits dem ASCII-Code und
    - die ersten 8 Bits der ASCII-Erweiterung ISO 8859-1 (Latin 1)



UTF


Unicode
0x000000-0x10FFFF
UTF 32
0x000000-0x10FFFF
UTF 16
0x000000-0x00FFFF
0x010000-0x10FFFF
mit v = u-1
UTF 8
0x000000-0x00007F
0x000080-0x0007FF
0x000800-0x00FFFF
0x010000-0x10FFFF

UTF

- Zur Kodierung von Unicode-Zeichen wird meistens das UTF "Universal Transformation Format" verwendet
- UTF-32 kodiert jedes Unicode-Zeichen mit 32 Bits, indem es die 21 Unicode Bits mit Nullen auffüllt
- UTF-16 kodiert alle Bits der Basic Multilingual Plane (BMP) mit 16 Bits, nur für die anderen Ebenen werden 32 Bits benötigt
- UTF-8 kodiert die ersten 7 Unicode Bits (entspricht ASCII) mit 8 Bits, die ersten 11 Unicode Bits mit 16 Bits, usw.
- Ein UTF-8 kodierter Text, der nur ASCII Zeichen enthält, ist demnach vollständig mit ASCII kompatibel
- UTF-8 ist heutzutage (besonders im Internet) weit verbreitet (Quasi-Standard der Zeichenkodierung)