

---

# — Informatik I —

## Modul 2: Rechnerarithmetik (1)



Universität  
Zürich<sup>UZH</sup>



---

## Modul 2: Rechnerarithmetik (1)

- **Zahlensysteme**
- Zahlendarstellung
- Zeichendarstellung

# Rechnerarithmetik

- Rechnerarithmetik soll als Beispiel vorbereitet werden, wie größere Informationseinheiten verarbeitet werden.
  - Hierzu werden zunächst die formalen Grundlagen erarbeitet:
    - Zahlensysteme
    - Zahlendarstellungen
    - Zeichendarstellungen
    - Boole'sche Algebra (Modul 3)
- bis 15.11.2011**
- Dann werden Schaltnetze und –werke behandelt (Modul 3, 4).
  - Danach werden Verfahren und Schaltungen zur Implementierung der vier Grundrechenarten in einem Rechner vorgestellt (Modul 5).
  - Abschließend wird die Funktion einer arithmetisch logischen Einheit (ALU) eines Rechners besprochen (Modul 5).

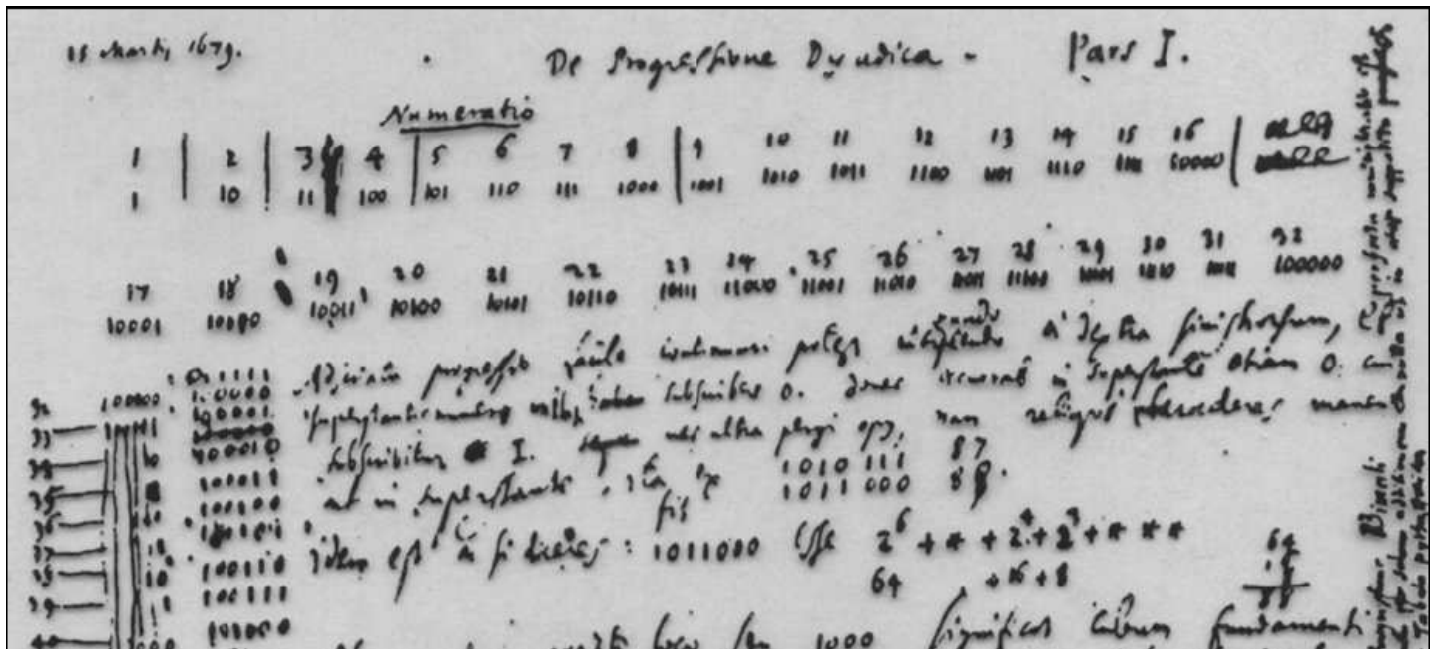
**ifi ab 16.11.2011**

## Zahlensysteme – Römische Zahlen

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

# Zahlensysteme – Leibniz'sche Dualzahlen

- Leibniz-Traktat aus dem Jahre 1679
- Vermutlich kommen Ideen zum Dualzahlensystem aus China



## Formale Grundlagen

- Menschen rechnen gewöhnlich im Dezimalzahlensystem.
- Rechner rechnen gewöhnlich im Dualzahlensystem.

→ Eine Konvertierung ist erforderlich

- Daneben werden weitere Zahlensysteme wie Oktalzahlensystem oder Hexadezimalzahlensystem (eigentlich: Sedezimal) zur kompakteren Darstellung der sehr langen Dualzahlen verwendet.

→ Es ist notwendig, die Zusammenhänge und mathematischen Grundlagen dieser Zahlensysteme zu verstehen.

# Zahlensysteme (1)

- Gängigste Form: Stellenwertsysteme

- Zahlendarstellung in Form einer Reihe von Ziffern  $z_i$ , wobei das Dezimalkomma (-punkt) rechts von  $z_0$  plaziert sei:

$$z_n z_{n-1} \dots z_1 z_0 , z_{-1} z_{-2} \dots z_{-m} \quad \text{z.B. } 1234,567$$

- Jeder Position  $i$  der Ziffernreihe ist ein Stellenwert zugeordnet, der eine Potenz  $b^i$  der Basis  $b$  des Zahlensystems ist.
- Der Wert  $X_b$  der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen  $z_i b^i$ :

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \dots + z_1 b + z_0 + z_{-1} b^{-1} + \dots + z_{-m} b^{-m} = \sum_{i=-m}^n z_i b^i$$

# Zahlensysteme (2)

- Interessante Zahlensysteme in der Informatik

b	Zahlensystem	Zahlenbezeichnung
2	Dualsystem	Dualzahl
8	Oktalsystem	Oktalzahl
10	Dezimalsystem	Dezimalzahl
16	Hexadezimalsystem (Sedezimalsystem)	Hexadezimalzahl (Sedezimalzahl)

- Hexadezimalsystem: Die „Ziffern“ 10 bis 15 werden mit den Buchstaben A bis F dargestellt.
- Dualsystem: Wichtigstes Zahlensystem im Rechner
- Oktal- und Hexadezimalsystem: Leicht ins Dualsystem umwandelbar, besser zu verstehen als lange 0-1-Kolonnen.

# Zahlensysteme (3)

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

## Der Euklidische Algorithmus

- Umwandlung vom Dezimalsystem in ein System zur Basis  $b$
- 1. Methode: Euklidischer Algorithmus:  
$$Z = z_n 10^n + z_{n-1} 10^{n-1} + \dots + z_1 10 + z_0 + z_{-1} 10^{-1} + \dots + z_{-m} 10^{-m}$$
$$= y_p b^p + y_{p-1} b^{p-1} + \dots + y_1 b + y_0 + y_{-1} b^{-1} + \dots + y_{-q} b^{-q}$$
Die Ziffern werden sukzessive, beginnend mit der höchstwertigen Ziffer, berechnet.
- 1. Schritt: Berechne  $p$  gemäß der Ungleichung  $b^p \leq Z < b^{p+1}$  (setze  $i = p$ )
- 2. Schritt: Ermittle  $y_i$  und den Rest  $R_i$  durch Division von  $Z_i$  durch  $b^i$ :  $y_i = Z_i \text{ div } b^i$ ;  $R_i = Z_i \text{ mod } b^i$ ;
- 3. Schritt: Wiederhole 2. Schritt für  $i = p-1, \dots$  und ersetze dabei nach jedem Schritt  $Z$  durch  $R_i$ , bis  $R_i = 0$  oder bis  $b^i$  (und damit der Umrechnungsfehler) gering genug ist.

# Beispiel

Umwandlung von  $15741,233_{10}$  ins Hexadezimalsystem:

1. Schritt:  $16^3 \leq 15741,233 < 16^4 \rightarrow$  höchste Potenz  $16^3$
2. Schritt:  $15741,233 : 16^3 = 3$  Rest 3453,233
3. Schritt:  $3453,233 : 16^2 = D$  Rest 125,233
4. Schritt:  $125,233 : 16 = 7$  Rest 13,233
5. Schritt:  $13,233 : 1 = D$  Rest 0,233
6. Schritt:  $0,233 : 16^{-1} = 3$  Rest 0,0455
7. Schritt:  $0,0455 : 16^{-2} = B$  Rest 0,00253
8. Schritt:  $0,00253 : 16^{-3} = A$  Rest 0,000088593
9. Schritt:  $0,000088593 : 16^{-4} = 5$  Rest 0,000012299  
( $\rightarrow$  Fehler)

$\rightarrow 15741,233_{10} \approx 3D7D,3BA5_{16}$

## Horner Schema

- Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis  $b$
- 2. Methode: Abwandlung des Horner Schemas
- Hierbei müssen der ganzzahlige und der gebrochene Anteil getrennt betrachtet werden:
- Umwandlung des ganzzahligen Anteils:
- Eine ganze Zahl  $X_b = \sum_{i=0}^n z_i b^i$  kann durch fortgesetztes Ausklammern auch in folgender Form geschrieben werden:

$$X_b = (((...(((y_n b + y_{n-1}) b + y_{n-2}) b + y_{n-3}) b \dots ) b + y_1) b + y_0$$

# Horner Schema: Beispiel

- Die gegebene Dezimalzahl wird sukzessive durch die Basis  $b$  dividiert.
- Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl  $X_b$  in der Reihenfolge von der niedrigstwertigen zur höchstwertigen Stelle.

Wandle  $15741_{10}$  ins Hexadezimalsystem um:

$$15741_{10} : 16 = 983 \quad \text{Rest } 13 \quad (D_{16})$$

$$983_{10} : 16 = 61 \quad \text{Rest } 7 \quad (7_{16})$$

$$61_{10} : 16 = 3 \quad \text{Rest } 13 \quad (D_{16})$$

$$3_{10} : 16 = 0 \quad \text{Rest } 3 \quad (3_{16})$$

$$\rightarrow 15741_{10} = 3D7D_{16}$$

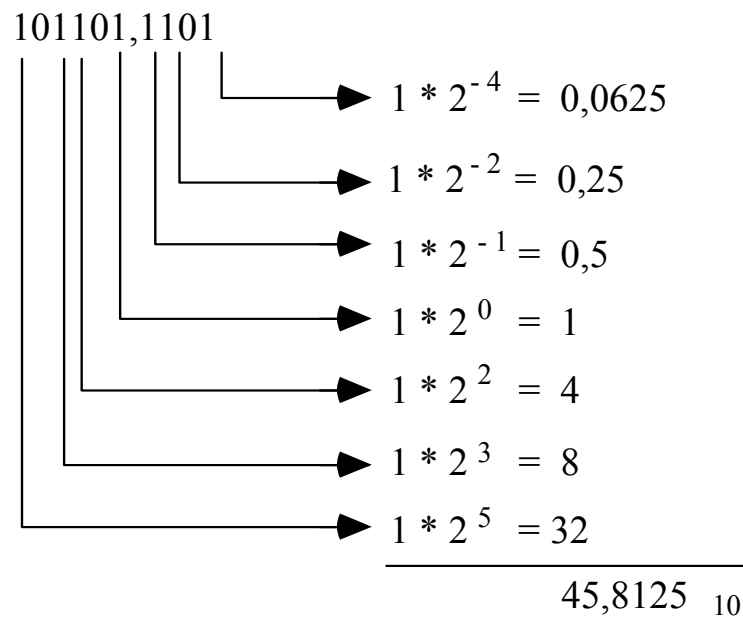
## Umwandlung: Basis $b \rightarrow$ Dezimalsystem

- Die Werte der einzelnen Stellen der umzuwandelnden Zahl werden in dem Zahlensystem, in das umgewandelt werden soll, dargestellt und nach der Stellenwertgleichung aufsummiert.
- Der Wert  $X_b$  der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen  $z_i b^i$ :

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \dots + z_1 b + z_0 + z_{-1} b^{-1} + \dots + z_{-m} b^{-m} = \sum_{i=-m}^n z_i b^i$$

# Beispiel

Konvertiere  $101101,1101_2$  ins Dezimalsystem



## Umwandlung beliebiger Stellenwertsysteme

- Man wandelt die Zahl ins Dezimalsystem um und führt danach mit Methode 1 oder 2 die Wandlung ins Zielsystem durch.
- Spezialfall:
  - Ist eine Basis eine Potenz der anderen Basis, können einfach mehrere Stellen zu einer Ziffer zusammengefasst werden oder eine Stelle kann durch eine Folge von Ziffern ersetzt werden.

- Wandlung von  $0110100,110101_2$  ins Hexadezimalsystem

- $2^4 = 16 \Rightarrow 4$  Dualstellen  $\rightarrow 1$  Hexadezimalstelle

dual

0110100,110101



00110100,11010100



hexadezimal

3 4 , D 4

Ergänzen von Nullen zur  
Auffüllung auf Vierergruppen



# Modul 2: Rechnerarithmetik (1)

---

- ❑ Zahlensysteme
- ❑ Zahlendarstellung
- ❑ Zeichendarstellung

## Darstellung negativer Zahlen

---

- ❑ Für die Darstellung negativer Zahlen in Rechnern werden vier verschiedene Formate benutzt :
- ❑ Darstellung mit Betrag und Vorzeichen
- ❑ Stellenkomplement-Darstellung (Einerkomplement-Darstellung)
- ❑ Zweierkomplement-Darstellung
- ❑ Offset-Dual-Darstellung / Exzeß-Darstellung

# Darstellung mit Betrag und Vorzeichen

- Eine Stelle wird als Vorzeichenbit benutzt.
- Ist das am weitesten links stehende Bit (MSB, most significant bit):
  - MSB = 0 → positive Zahl
  - MSB = 1 → negative Zahl

Beispiel:

$$\begin{array}{rcl} 0001\ 0010 & = & +18 \\ 1001\ 0010 & = & -18 \end{array}$$

## □ Nachteile:

- Bei Addition und Subtraktion müssen die Vorzeichen der Operanden gesondert betrachtet werden.
- Es gibt zwei Repräsentationen der Zahl 0 (mit positivem und mit negativem Vorzeichen)

# Stellenkomplement / Einerkomplement

- Stellenkomplement der entsprechenden positiven Zahl.
- Um eine Zahl zu negieren, wird jedes Bit der Zahl komplementiert.
- Dies entspricht dem Einerkomplement:

*Komplementbildung*

$$z_{ek} = (2^n - 1) - z$$

Bsp:  $4 = 0100_2 \rightarrow -4 = 1011_{ek}$

$$-4 = 2^4 - 1 - 4 = 11_{10} = 1011_2$$

- Negative Zahlen sind wiederum durch ein gesetztes Bit in der ersten Stelle charakterisiert.
- Vorteil gegenüber der Darstellung mit Vorzeichenbit:
  - Erste Stelle bei Addition und Subtraktion muß nicht gesondert betrachtet werden.
  - Aber: **Es gibt weiterhin zwei Darstellungen der Null**



# Beispiel

Die Zahl  $-77_{10}$  soll mit 8 Bit dargestellt werden

$$77_{10} = 0100\ 1101_2$$

Bitweise komplementieren

Mit Vorzeichenbit :  $-77 = 1100\ 1101_2$

Einerkomplement :  $-77 = 1011\ 0010_2$

Zweierkomplement :  $-77 = 1011\ 0011_2$

Addition von 1

## Offset-Dual- (Exzeß-)Darstellung

- Wird hauptsächlich bei der Exponenten-Darstellung von Gleitkommazahlen benutzt.
- Die Darstellung einer Zahl erfolgt in Form ihrer **Charakteristik (bias)**.
- Der gesamte Zahlenbereich wird durch Addition einer Konstanten (Exzeß, Offset) so nach oben verschoben, daß die kleinste (negative) Zahl die Darstellung **0...0** erhält.
- Bei  $n$  Stellen ist der Offset  $2^{n-1}$
- Der Zahlenbereich ist hier auch asymmetrisch.

# Zusammenfassung der Möglichkeiten

Darstellung mit				
Dezimalzahl	Betrag + Vorzeichen	Einerkomplement	Zweierkomplement	Charakteristik
-4	- - -	- - -	1 0 0	0 0 0
-3	1 1 1	1 0 0	1 0 1	0 0 1
-2	1 1 0	1 0 1	1 1 0	0 1 0
-1	1 0 1	1 1 0	1 1 1	0 1 1
0	1 0 0, 0 0 0	1 1 1, 0 0 0	0 0 0	1 0 0
1	0 0 1	0 0 1	0 0 1	1 0 1
2	0 1 0	0 1 0	0 1 0	1 1 0
3	0 1 1	0 1 1	0 1 1	1 1 1

## Fest- und Gleitkommazahlen

- Zahlendarstellung auf dem Papier:

Ziffern                    0 .. 9

Vorzeichen                + -

Komma (Punkt)            , .

- Zahlendarstellung im Rechner:

Binärziffern    0, 1

→ spezielle Vereinbarungen für die Darstellung von Vorzeichen und Komma/Punkt im Rechner sind erforderlich

- Darstellung des **Vorzeichens**:

– Wurde im vorigen Abschnitt behandelt

- Darstellung des **Kommas** mit zwei Möglichkeiten:

– Festkommadarstellung

– Gleitkommadarstellung

# Festkomma-Zahlen (1)

- Vereinbarung:
  - Das Komma sitzt innerhalb des Maschinenwortes, das eine Dualzahl enthalten soll, an einer festen Stelle.
- Meist setzt man das Komma hinter die letzte Stelle.
- Andere Zahlen können durch entsprechende Maßstabsfaktoren in die gewählte Darstellungsform überführt werden.
- Negative Zahlen:
  - Meist Zweierkomplement-Darstellung.
- Festkomma-Darstellungen werden heute hardwareseitig nicht mehr verwendet, jedoch bei der Ein- oder Ausgabe!

# Festkomma-Zahlen (2)

- Datentyp "integer" (Ganzzahlen) ist ein spezielles Festkommaformat.
- Manche Programmiersprachen erlauben die Definition von Ganzzahlen unterschiedlicher Länge.
- Beispiel "C": "short int", "int", "long int", "unsigned"

Datentyp	DEC-VAX (einer der Urahnen)		IBM-PC, Apple Macintosh	
	Anzahl der Bits	Zahlenbereich	Anzahl der Bits	Zahlenbereich
short int	16	$-2^{15} .. 2^{15}-1$	16	$-2^{15} .. 2^{15}-1$
int	32	$-2^{31} .. 2^{31}-1$	16	$-2^{15} .. 2^{15}-1$
long int	32	$-2^{31} .. 2^{31}-1$	32	$-2^{31} .. 2^{31}-1$

# Gleitkomma-Darstellung (1)

- Zur Darstellung von Zahlen, die betragsmäßig sehr groß oder sehr klein sind, verwendet man die **Gleitkommadarstellung**.
- Sie entspricht einer halblogarithmischen Form

$$X = \pm \text{Mantisse} \cdot b^{\text{Exponent}}$$

- Die Basis  $b$  ist für eine bestimmte Gleitkomma-Darstellung fest (meist 2 oder 16) und braucht damit nicht mehr explizit repräsentiert zu werden.
- Gleitkommazahlen werden meist *nicht* im Zweierkomplement, sondern mit **Betrag** und **Vorzeichen** dargestellt.

# Gleitkomma-Darstellung (2)

- Bei der **Mantisse** ist die Lage des Kommas wieder durch Vereinbarung festgelegt (meist links vom MSB).
- Der **Exponent** ist eine ganze Zahl, die in Form ihrer Charakteristik dargestellt wird.
- Für die Charakteristik und die Mantisse wird im Rechner ein **festen Anzahl von Speicherstellen** festgelegt.
- Die Länge der Charakteristik  $y-x$  bestimmt die Größe des Zahlenbereichs.
- Die Länge der Mantisse  $x$  legt die Genauigkeit der Darstellung fest.



$$\text{Dezimalzahl} = (-1)^{Vz} * (0, \text{Mantisse}) * b^{\text{Exponent}}$$

$$\text{Exponent} = \text{Charakteristik} - b^{(y-1) - x}$$

# Normalisierung

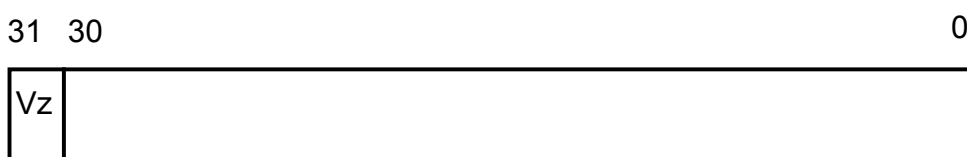
- ❑ Legt man für die Zahl 0 ein spezielles Bitmuster fest, ist die erste Stelle der Mantisse in normalisierter Form immer gleich 1.
- ❑ Die erste Stelle der Mantisse braucht im Maschinenformat gar nicht erst dargestellt zu werden, d.h. (0,1 . . . . .)
- ❑ Man spart ein Bit bei der Speicherung oder gewinnt bei gleichem Speicherbedarf ein Bit an Genauigkeit.
- ❑ Bei arithmetischen Operationen und bei der Konversion in andere Darstellungen darf diese Stelle natürlich nicht vergessen werden.

## Beispiel (1)

3 verschiedene Maschinenformate mit je 32 Bit und  $b = 2$ .

Die Zahl  $7135_{10}$  wird in jedem dieser Formate dargestellt.

a) Festkommadarstellung mit Zweierkomplement

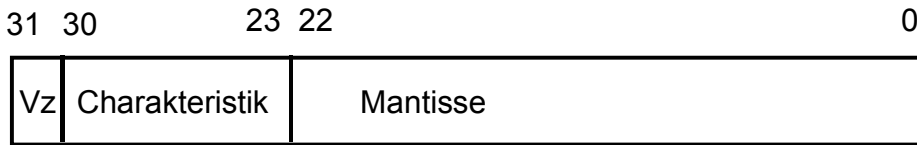


$$0\ 000\ 0000\ 0000\ 0000\ 0001\ 1011\ 1101\ 1111_2 = 0000\ 1BDF_{16}$$



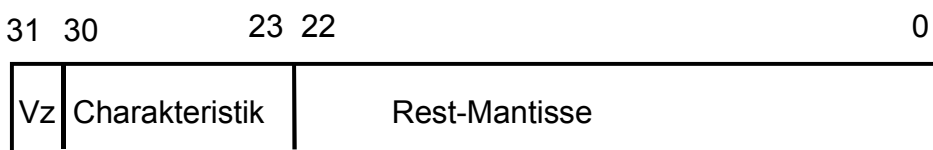
## Beispiel (2)

b) Gleitkommadarstellung, normalisiert:



$$0 \ 100 \ 0110 \ 1 \ 110 \ 1111 \ 0111 \ 1100 \ 0000 \ 0000_2 = 46EF \ 7C00_{16}$$

c) Gleitkommadarstellung, normalisiert, erste "1" implizit:



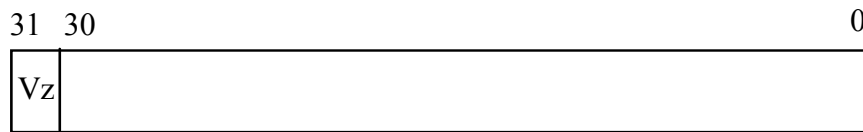
$$0 \ 100 \ 0110 \ 1 \ 101 \ 1110 \ 1111 \ 1000 \ 0000 \ 0000_2 = 46DE \ F800_{16}$$

## Darstellbarer Zahlenbereich (1)

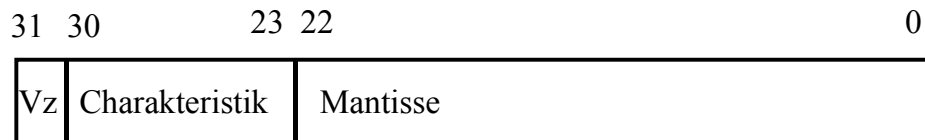
- Die Anzahl darstellbarer Zahlen (Bitkombinationen) ist zwar in allen drei Fällen gleich ( $2^{32}$ )
- Der Bereich und damit die Dichte darstellbarer Zahlen auf dem Zahlenstrahl ist aber sehr unterschiedlich.

## Darstellbarer Zahlenbereich (2)

- Format a: Zahlen zwischen  $-2^{31}$  und  $2^{31}-1$



- Format b:



negative Zahlen:  $-(1-2^{-23}) \cdot 2^{127} \dots -0,5 \cdot 2^{-128}$ ,

positive Zahlen  $0,5 \cdot 2^{-128} \dots (1-2^{-23}) \cdot 2^{127}$ ,

und *Null*

## Darstellbarer Zahlenbereich (3)

- Format c: normalisierte Gleitkommadarstellung

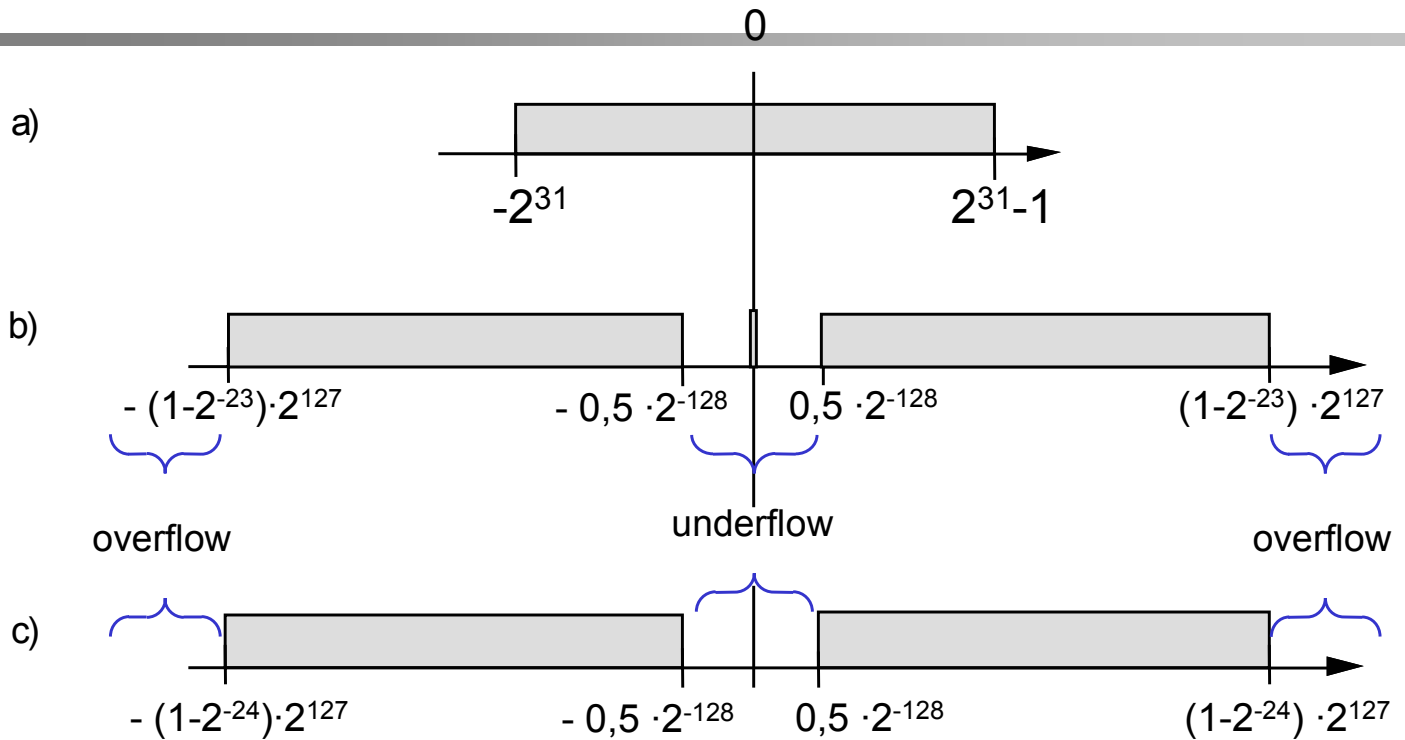


negative Zahlen:  $-(1-2^{-24}) \cdot 2^{127} \dots -0,5 \cdot 2^{-128}$

positive Zahlen  $0,5 \cdot 2^{-128} \dots (1-2^{-24}) \cdot 2^{127}$

**Die Null kann nicht dargestellt werden!**

## Darstellbarer Zahlenbereich (4)



## Charakteristische Zahlen

- Um verschiedene Gleitkommadarstellungen miteinander vergleichen zu können, definiert man drei charakteristische Zahlen:
- **maxreal** ist die größte darstellbare normalisierte positive Zahl
- **minreal** ist die kleinste darstellbare normalisierte positive Zahl
- **smallreal** ist die kleinste Zahl, die man zu 1 addieren kann, um einen von 1 verschiedenen Wert zu erhalten.



# Beispiel

- Das Assoziativgesetz  $(x + y) + z = x + (y + z)$  gilt selbst dann nicht unbedingt, wenn kein overflow oder underflow auftritt.

z.B.:  $x = 1; y = z = \text{smallreal}/2$

$$\begin{aligned}(x + y) + z &= (1 + \text{smallreal}/2) + \text{smallreal}/2 \\ &= 1 + \text{smallreal}/2 \\ &= 1\end{aligned}$$

$$\begin{aligned}x + (y + z) &= 1 + (\text{smallreal}/2 + \text{smallreal}/2) \\ &= 1 + \text{smallreal} \\ &\neq 1\end{aligned}$$

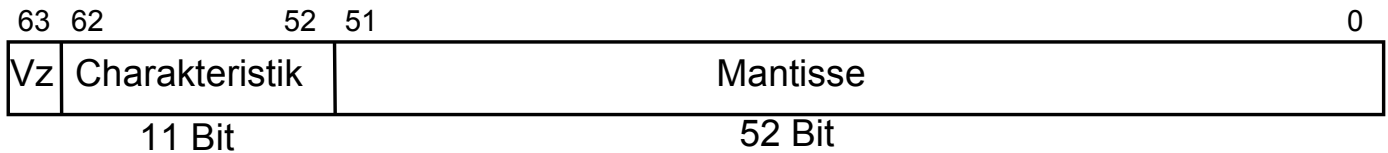
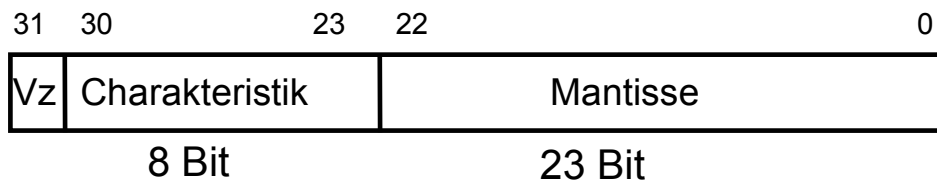
Hinweis: **smallreal** ist die kleinste Zahl, die man zu 1 addieren kann, um einen von 1 verschiedenen Wert zu erhalten!

## Problematik unterschiedlicher Definitionen

- Es existieren beliebig viele Möglichkeiten, selbst mit einer festen Wortbreite unterschiedliche Gleitkommaformate zu definieren (unterschiedliche Basis  $b$ , Darstellung der Null, Anzahl der Stellen für Charakteristik und Mantisse).
- Es existierten (bis Mitte der 80er Jahre) viele verschiedene, herstellerabhängige Formate
- Man konnte mit dem gleichen Programm auf unterschiedlichen Rechnern sehr unterschiedliche Ergebnisse erhalten!
- Normierung erforderlich

# Normierung: IEEE-Standard (1)

- IEEE-P 754-Floating-Point-Standard
- In vielen Programmiersprachen lassen sich Gleitkomma-Zahlen mit verschiedener Genauigkeit darstellen
  - z.B. in C: float, double, long double
- Der IEEE Standard definiert mehrere Darstellungsformen
  - IEEE single: 32 Bit
  - IEEE double: 64 Bit
  - IEEE extended: 80 Bit



# Normierung: IEEE-Standard (2)

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

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

# Modul 2: Rechnerarithmetik (1)

---

- Zahlensysteme
- Zahlendarstellung
- Zeichendarstellung

## ASCII Code zur Darstellung von Zeichen (1)

---

- ASCII-Code (American Standard for Coded Information Interchange) ist eine festgelegte Abbildungsvorschrift (Norm) zur binären Kodierung von Zeichen.
  - Der ASCII-Code umfasst Klein-/Großbuchstaben des lateinischen Alphabets, (arabische) Ziffern und viele Sonderzeichen.
  - Die Kodierung erfolgt in einem Byte (8 Bits), so daß mit dem ASCII-Code 256 verschiedene Zeichen dargestellt werden können.
  - Da das erste Bit nicht vom Standard-ASCII-Code genutzt wird, können im Standard-ASCII-Code nur 128 Zeichen dargestellt werden.
  - Unterschiedliche, speziell normierte, ASCII-Code-Erweiterungen nutzen das erste Bit, um weitere 128 Zeichen darstellen zu können.

## ASCII Code zur Darstellung von Zeichen (2)

### ASCII-Code zu den darstellbaren Zeichen

Zeichen	Dez.	Binär	Hexa	Oktal	Zeichen	Dez.	Binär	Hexa	Oktal
!	33	0010 0001	21	041	P	80	0101 0000	50	120
"	34	0010 0010	22	042	Q	81	0101 0001	51	121
#	35	0010 0011	23	043	R	82	0101 0010	52	122
\$	36	0010 0100	24	044	S	83	0101 0011	53	123
%	37	0010 0101	25	045	T	84	0101 0100	54	124
&	38	0010 0110	26	046	U	85	0101 0101	55	125
'	39	0010 0111	27	047	V	86	0101 0110	56	126
(	40	0010 1000	28	050	W	87	0101 0111	57	127
)	41	0010 1001	29	051	X	88	0101 1000	58	130

## ASCII Code zur Darstellung von Zeichen (3)

- Zur Speicherung von Texten werden einzelne Bytes, die jeweils immer ein Zeichen kodieren, einfach hintereinander abgespeichert, so daß man eine Zeichenkette (*String*) erhält.
- Um das Ende der Zeichenkette zu identifizieren, werden (in den Programmiersprachen) unterschiedliche Verfahren verwendet.
  - Die Länge der Zeichenkette wird im ersten bzw. in den ersten Bytes vor der eigentlichen Zeichenkette gespeichert.  
Dieses Verfahren benutzt z. B. die Programmiersprache PASCAL.
  - Das Ende der Zeichenkette wird durch ein besonderes, nicht darzustellendes Zeichen gekennzeichnet.  
So verwendet z.B. die Programmiersprache C/C++ ein 0-Byte (Byte, in dem alle Bits 0 sind), um das Ende einer Zeichenkette zu kennzeichnen.



# Unterscheidung von Ziffern und Zeichen

---

- *Ziffer als ASCII-Code* → *Angabe des Zeichens (Ziffer) in Hochkomma:*

'0': 00110000 (Dezimal 48)  
'4': 00110100 (Dezimal 52)  
'5': 00110101 (Dezimal 53)  
'8': 00111000 (Dezimal 56)

- *Ziffer als numerischer Wert* → *Angabe einer Ziffer (ohne Hochkomma):*

0: 00000000 (Dezimal 0)  
4: 00000100 (Dezimal 4)  
5: 00000101 (Dezimal 5)  
8: 00001000 (Dezimal 8)

## Beispiel: Speicherung von Ziffern und Zeichen

---

Angabe	Dezimaler ASCII-Wert	Dualdarstellung im Rechner
'a'	97	01100001
'W'	87	01010111
'*'	42	00101010
'9'	57	00111001

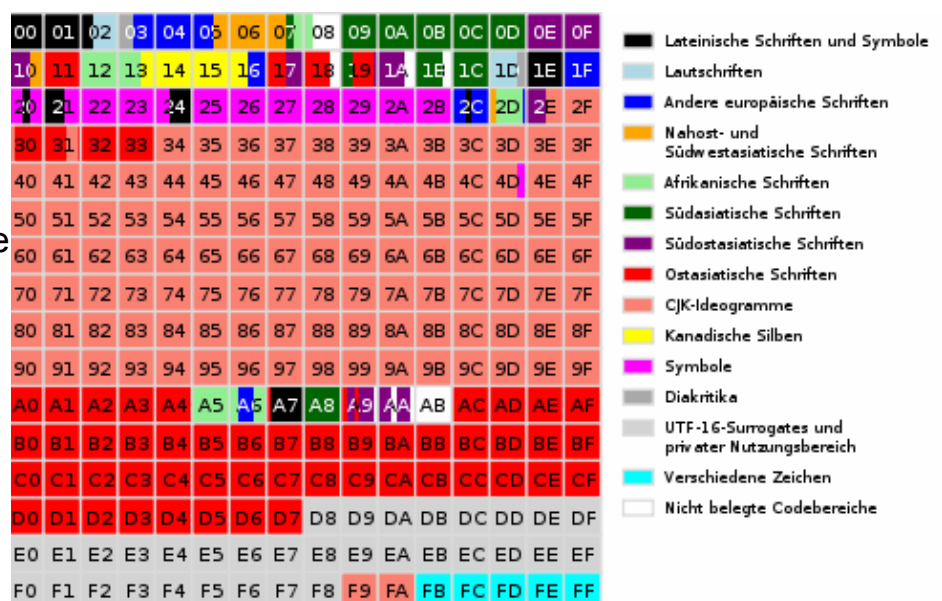
# Unicode zur Darstellung von Zeichen (1)

- ❑ Der ASCII-Code mit 256 Zeichen ist sehr begrenzt.
- ❑ Unicode für Zeichen oder Elemente praktisch aller bekannten Schriftkulturen und Zeichensysteme kodierbar.
- ❑ Zeichenwerte der Zeichen bis Unicode Version 3.0 (September 1999) wurden ausschließlich durch eine zwei Byte lange Zahl ausgedrückt.
  - Auf diese Weise lassen sich bis zu 65 536 verschiedene Zeichen unterbringen (2 Byte = 16 Bit =  $2^{16}$  Kombinationsmöglichkeiten).
  - *Bezeichnung des Zwei-Byte-Schemas: Basic Multilingual Plane (BMP)*
- ❑ In Version 3.1 (März 2001) sind 94.140 Zeichen aufgenommen, wobei die Zwei-Byte-Grenze durchbrochen wurde.
  - Das Zwei-Byte-Schema wird deshalb von einem Vier-Byte-Schema abgelöst.



# Unicode zur Darstellung von Zeichen (2)

- ❑ In Version 6.0 (Oktober 2010) sind 109.449 Zeichen enthalten.
  - Unicode in 17 Bereiche (planes) gegliedert, jeweils á 65.536 Zeichen
  - Basic Multilingual Plane: hauptsächliche Schriftsysteme
  - Supplementary Multilingual Plane: historische Schriftsysteme
  - Supplementary Ideographic Plane: Chinesische, Japanische und Koreanische Schrift
  - Supplementary Special-purpose Plane: Private Use Area-A und -B



# Unicode zur Darstellung von Zeichen (3)

- Die Notation lautet
  - &#0000; für dezimale Notation bzw.
  - &#x0000; für die hexadezimale Notation,wobei das 0000 die Unicode-Nummer des Zeichens darstellt.
- Formate für Speicherung und Übertragung unterschiedlich:
  - UTF-8 (Unicode Transformation Format) – Internet und in Betriebssystemen
  - UTF-16 – Zeichencodierung in Java
- Unicode wird in ostasiatischen Ländern kritisiert.
  - Schriftzeichen verschiedener nicht verwandter Sprachen sind vereinigt
  - Vor allem in Japan konnte sich der Unicode kaum durchsetzen
  - Japan mit zahlreichen Alternativen zu Unicode wie etwa der Mojikyō-Standard
  - Antike Texte in Unicode aufgrund der Vereinheitlichung ähnlicher CJK-Schriftzeichen (chinesisch, japanisch, koreanisch) nicht originalgetreu wiederzugeben

# BCD-Code zur Darstellung von Zeichen

- BCD (Binary Coded Decimals) kodieren binär Zahlen Ziffern.
- Für jede Dezimalziffer werden mindestens vier, manchmal auch acht Bits verwendet.
  - Die Ziffern werden nacheinander immer durch ihren Dualwert angegeben.
  - Diese speicherplatzverschwendende Art der Speicherung von Dezimalzahlen erleichtert aber manche Anwendungen.
  - Anwendungsbereiche:
    - Rechnen im Dezimalsystem
    - Speichern von Dezimalzahlen (Telefonnummern u.ä.)
    - Ansteuerung von LCD-Anzeigen, um Dezimalziffern einzeln anzuzeigen.

Dezimalzahl	Dualzahl	Duale BCD-Darstellung
294	100100110	0010.1001.0100 2 9 4
16289	11111110100001	0001.0110.0010.1000.1001 1 6 2 8 9

# Duale Größenangaben (1)

## Maßeinheiten für Bytes

Maßeinheit		Anzahl von Bytes	KBytes	MBytes
Byte		1		
Kilobyte (KByte)	$2^{10}$	1024	1	
Megabyte (MByte)	$2^{20}$	1.048.576	1024	1
Gigabyte (GByte)	$2^{30}$	1.073.741.824	1.048.576	1024
Terabyte (TByte)	$2^{40}$	1.099.511.627.776	1.073.741.824	1.048.576
Petabyte (PByte)	$2^{50}$	1.125.899.906.842.624	1.099.511.627.776	1.073.741.824
Exabyte (EByte)	$2^{60}$	1.152.921.504.606.846.976	1.125.899.906.842.624	1.099.511.627.776

# Duale Größenangaben (2)

- ❑ Kilo entspricht dem Faktor  $2^{10} = 1024$
- ❑ Mega entspricht dem Faktor  $2^{10} * 2^{10} = 1024 * 1024 = 1.048.576$
- ❑ Giga entspricht dem Faktor  $2^{10} * 2^{10} * 2^{10} = 2^{30}$
  
- ❑ Beispiel Festplattenherstellerangaben zur Speicherkapazität
  
- ❑ MB  $\neq$  MByte
  - Faktor hier  $10^3 = 1000$
  
- ❑ Folge:
  - 1 GB  $\neq$  1.073.741.824 Byte sondern 1.000.000.000 Byte
    - D.h., es „fehlen“ real 73 MegaByte!
  - 200 GB entsprechen damit nur 186 Gbyte!