Einführung in die Technische Informatik

Prof. Dr.-Ing. Stefan Kowalewski

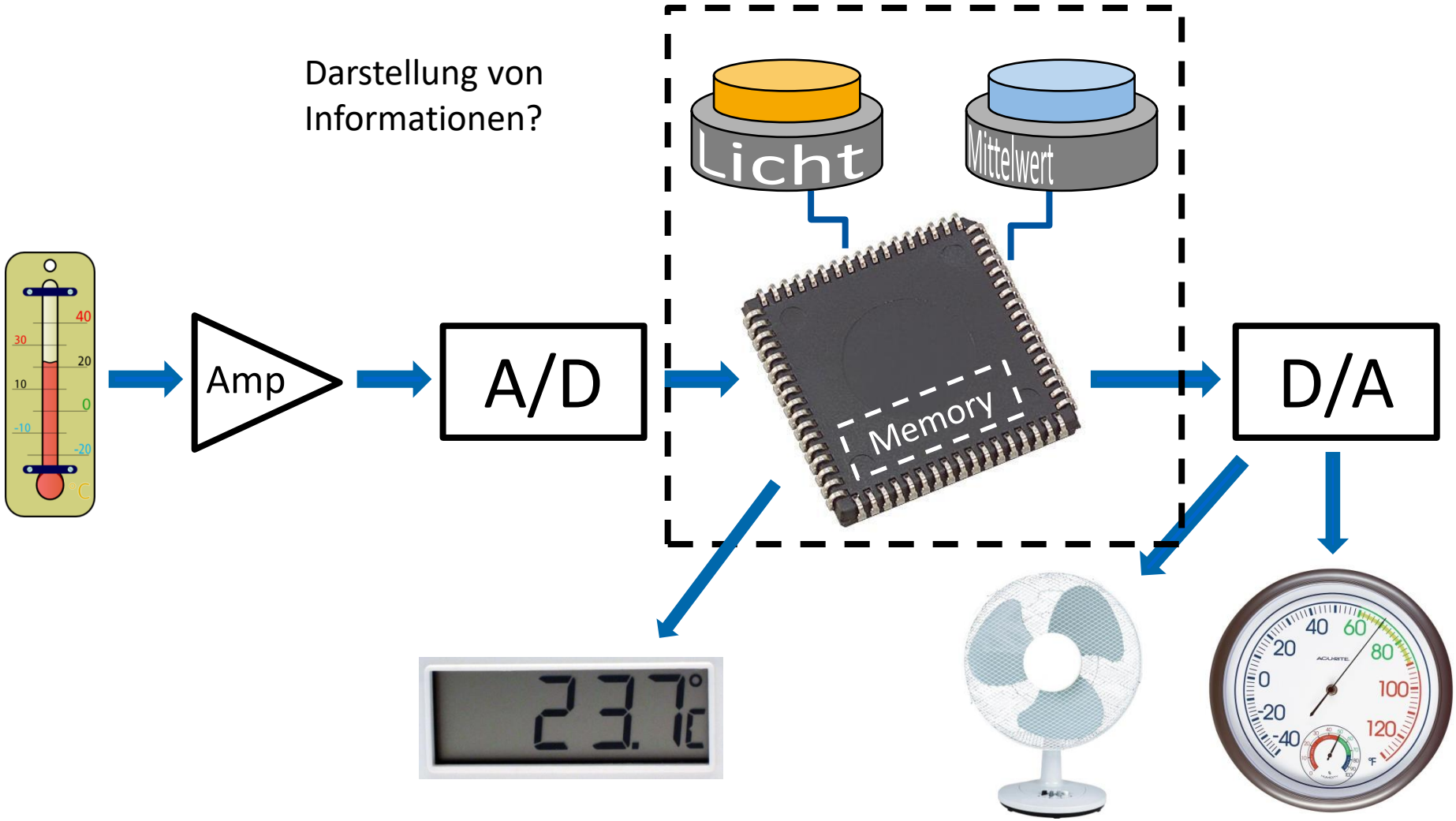
WS 22/23

Kapitel 1: Zahlendarstellung

Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Anwendungsbeispiel



Abschnitt 1.1

Natürliche Zahlen

- Zahlendarstellungen
- b-adische Darstellung natürlicher Zahlen

Zahlendarstellungen

Endliches Alphabet:

z.B. im Dezimalsystem: $\sum_{10} = \{0,1,2,3,4,5,6,7,8,9\}$

allgemein: $\sum_b = \{0,1,\dots,b-1\}$, wobei b **Basis** genannt wird

b-adische Zahlen mit **endlicher Wortlänge** n: $\sum_b^n$

Beispiel: $00456 \in \sum_{10}^5$

Wichtige Zahlensysteme in der Informatik:

Dual-/Binärsystem: $\sum_2 = \{0,1\}$

Oktalsystem: $\sum_8 = \{0,1,2,3,4,5,6,7\}$

Hexadezimalystem: $\sum_{16} = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Wahl der Basis

- Warum keine Basis 10?
  - Natürliche Darstellung für Finanztransaktionen, etwa € 1.20
  - Verbreitet in Wissenschaft, z.B. $6.6206 \cdot 10^{34}$

- Aber elektronische Bearbeitung schwierig
  - Speicherung anspruchsvoll, der ENIAC benutzte 10 Röhren pro Ziffer
  - Übertragung von 10 Signalniveaus auf einer Leitung benötigt hohe Präzision
  - Implementierung von Addition & Co ist schwierig

b-adische Darstellung natürlicher Zahlen

Sei $b \in N$ mit $b > 1$. Dann ist jede natürliche Zahl z mit $0 \leq z \leq b^n - 1$ (und $n \in N$) eindeutig als Wort der Länge n über $\sum_b^n$ darstellbar durch

$$z = \sum_{i=0}^{n-1} z_i b^i$$

mit $z_i \in \sum_b = \{0,1, \dots, b-1\}$ für $i = 0,1, \dots, n-1$.

Als vereinfachende Schreibweise ist dabei die folgende **Ziffernschreibweise** üblich:

$$z = (z_{n-1} z_{n-2} \dots z_1 z_0)_b$$

Wichtiger Spezialfall: $b = 2$ („**Binärdarstellung**“ natürlicher Zahlen)

Beispiel: Binäre Zahlendarstellung

- Betrachte $z = (0110)_2$
- Als natürliche Zahl interpretiert als:

$$z = \sum_{i=0}^{3} 2^i \cdot z_i = 2^0 \cdot 0 + 2^1 \cdot 1 + 2^2 \cdot 1 + 2^3 \cdot 0 = 6$$

- Größte darstellbare Zahl mit n Bits:

$$z_{max} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

- Beispiel: z ist 3-stellig $\Rightarrow z_{max} = (111)_2 = (7)_{10} = 2^3 - 1$

Beispiel: Umrechnung Dezimal- zu Binärsystem

47	: 2 =	23	Rest	1
23	: 2 =	11	Rest	1
11	: 2 =	5	Rest	1
5	: 2 =	2	Rest	1
2	: 2 =	1	Rest	0
1	: 2 =	0	Rest	1

$(47)_{10} = (101111)_2$

Horner-Schema

- Schema zur Umrechnung zwischen Zahlensystemen mit unterschiedlichen Basen

$$w = \sum_{i=0}^{n-1} z_i b^i = z_0 + b \cdot (z_1 + b \cdot (z_2 + \cdots \left(b \cdot \left(z_{n-3} + b \cdot (z_{n-2} + b \cdot z_{n-1})\right) \ldots\right))$$

- Ermittle Sequenz durch Division mit Rest:

$$\frac{w}{b} = \underbrace{z_1 + b \cdot (z_2 + \cdots \left(b \cdot \left(z_{n-3} + b \cdot (z_{n-2} + b \cdot z_{n-1})\right) \ldots\right))}_{w_1} \quad \text{Rest} \quad z_0$$

$$\frac{w_1}{b} = \underbrace{z_2 + \cdots \left(b \cdot \left(z_{n-3} + b \cdot (z_{n-2} + b \cdot z_{n-1})\right) \ldots\right)}_{w_2} \quad \text{Rest} \quad z_1$$

$$\frac{w_{n-2}}{b} = \underbrace{z_{n-1}}_{w_{n-1}} \quad \text{Rest} \quad z_{n-2}$$

$$\frac{w_{n-1}}{b} = 0 \quad \text{Rest} \quad z_{n-1}$$

Beispiel: Horner-Schema

Beispiel: $(2595)_{10}$ nach Basis 11

$$\frac{(2595)_{10}}{11} = (235)_{10} \qquad \text{Rest: } (10)_{10} = (A)_{11}$$

$$\frac{(235)_{10}}{11} = (21)_{10} \qquad \text{Rest: } (4)_{10} = (4)_{11}$$

$$\frac{(21)_{10}}{11} = (1)_{10} \qquad \text{Rest: } (10)_{10} = (A)_{11}$$

$$\frac{(1)_{10}}{11} = (0)_{10} \qquad \text{Rest: } (1)_{10} = (1)_{11}$$

$$\Rightarrow (2595)_{10} = (1A4A)_{11}$$

Addition

- mod-Operation („modulo“): $a \bmod b$ ist der Rest der Division von a durch b
- Resultat von $x + y$ ist definiert für n Bits als $(x + y) \bmod 2^n$
- Beispiel:

$$\begin{array}{rl} & 0011 \\ + & 0010 \\ \hline & 0101 \end{array} \qquad \begin{array}{l} 3 = 2^0 + 2^1 \\ 2 = 2^1 \\ \hline 5 = 2^0 + 2^2 \end{array}$$

- Überlauf eines jeden Bits ergibt Übertrag

Überlauf bei Addition

Beispiel: Was geschieht bei $15 + 2$?

$$\begin{array}{r} 1111 \\ +\ 0010 \\ \hline 0001 \end{array} \qquad \begin{array}{l} 15 = 2^0 + 2^1 + 2^2 + 2^3 \\ 2 = 2^1 \\ \hline 1 = 2^0 \end{array} \Bigg\} \; 1 = (15 + 2) \bmod 2^4$$

Überläufe entstehen durch endliche Präzision. Die implementierte Addition ist daher „modular“, und das Resultat von $x + y$ bei n Bits ist definiert als $(x + y) \bmod 2^n$

Multiplikation

Schulmethode:

Sei x der Multiplikand, $y = (y_{n-1}, \ldots, y_0)$ der Multiplikator, dann ist

$$x \cdot y = x \cdot y_0 + x \cdot y_1 \cdot 2 + x \cdot y_2 \cdot 2^2 + \cdots + x \cdot y_{n-1} \cdot 2^{n-1}$$

$$= \sum_{i=0}^{n-1} x \cdot y_i \cdot 2^i$$

In der Praxis ist es sinnvoll, jeden Term der Form $x \cdot y_i \cdot 2^i$ zu addieren, sobald er generiert wurde:

$$x \cdot y = \left(\ldots\left(\left(x \cdot y_0 + x \cdot y_1 \cdot 2\right) + x \cdot y_2 \cdot 2^2\right) + \cdots\right) + x \cdot y_{n-1} \cdot 2^{n-1}$$

Abschnitt 1.2

Ganze Zahlen

- Darstellung ganzer Zahlen im Rechner
- Alternative Darstellungen ganzer Zahlen
- BCD-Code

Darstellung ganzer Zahlen im Rechner

Bisher wurden nur natürliche Zahlen betrachtet. Wie sollte man ganze Zahlen darstellen?
Der konzeptionell einfachste Ansatz:

Dualdarstellung wie bisher plus Vorzeichen-Bit
0 = +, 1 = -, Bsp. +5 = 0101, -5 = 1101

Nachteile: 0 hat zwei Darstellungen und
man braucht sowohl Addier- als auch Subtrahierwerk:

1.	*+x, +y*	*x+y*	Add.
2.	*-x,-y*	*-(x+y)*	Add.
3.	*+x, -y (x ≥ y)* *-x, +y (y ≥ x)*	*x-y* *y-x*	Subtr. Subtr.
4.	*+x, -y (x < y)* *-x, +y (y < x)*	*-(y-x)* *-(x-y)*	Subtr. Subtr.

Nachteile der vorzeichenlosen Version → Verwendung von $K_1(x)$ und $K_2(x)$

Darstellung ganzer Zahlen im Rechner

Wir arbeiten hauptsächlich auf der Menge $B = \{0,1\}$.
B^n beschreibt dann die Menge der n-stelligen Binärzahlen.

Sei $x = (x_{n-1} \dots x_0)_2 \in B^n$ eine n-stellige Binärzahl.

(i) $K_1(x) := (\overline{x_{n-1}}, \dots, \overline{x_0})_2$ heißt **Einer-Komplement** von x

(ii) $K_2(x) := (\overline{x_{n-1}}, \dots, \overline{x_0})_2 + 1 = K_1(x) + 1 \text{ (modulo) } 2^n$
heißt **Zweier-Komplement** von x

Beispiel: x = 10110010: $K_1(x) = 01001101$
$K_2(x) = 01001110$

Umrechnungsregeln

- Einerkomplement: Invertierung

Beispiel:

$$x = 10110010$$
$$K_1(x) = 01001101$$

- Zweierkomplement: Invertierung + 1

Beispiel:

$$x = 10110010$$
$$K_1(x) = 01001101$$
$$K_2(x) = 01001110$$

Beispiel: Vorzeichenbehaftete Zahlen mit 4 Bit

- Das MSB legt das Vorzeichen fest
- $(0???)_2$
- $(1???)_2$

  - Wie interpretieren wir die Bits bei den Fragezeichen?

Einerkomplement

- $(0???)_2$: wird interpretiert wie im vorzeichenlosen Fall
- $(1???)_2$: Bilde Einerkomplement K_1
- Beispiel:

$$(0010)_2 = +(010)_2 = (2)_{10}$$

$$(1010)_2 = \text{-}\mathrm{K}_1(1010)_2 = \text{-}(0101)_2 = \text{-}(5)_{10}$$

- $K_1(K_1(x)) = x$

Addition im Einerkomplement

Beispiel: $(3)_{10} + (-6)_{10} = -(3)_{10}$

$(3)_{10} = (0011)_2$
$-(6)_{10} = K_1(0110)_2 = (1001)_2$

$$\begin{array}{r} 0011 \\ +1001 \\ \hline 1100 \end{array}$$

$(1100)_2 = -K_1(1100)_2 = -(0011)_2$
$= -(3)_{10}$

Zweierkomplement

- $(0???)_2$: wird interpretiert wie im vorzeichenlosen Fall
- $(1???)_2$: Bilde Zweierkomplement K_2
- Beispiel:

$$(0010)_2 = +(010)_2 = (2)_{10}$$

$$(1010)_2 = \text{-}\mathrm{K}_2(1010)_2 = \text{-}(0101 + 1)_2 = \text{-}(6)_{10}$$

- $K_2(K_2(x)) = x$

Addition im Zweierkomplement

Beispiel: $(3)_{10} + (-6)_{10} = -(3)_{10}$

$(3)_{10} = (0011)_2$
$-(6)_{10} = K_2(0110)_2 = (1001 + 1)_2 = (1010)_2$

$$\begin{array}{r} 0011 \\ +1010 \\ \hline 1101 \end{array}$$

$(1101)_2 = -K_2(1101)_2 = -(0010 + 1)_2$
$= -(3)_{10}$

Darstellungsbereiche bei 4 Bit

- Vorzeichenlos:

  $(0)_{10} \cdots (15)_{10}$

- Einerkomplement:

  $-(7)_{10} \cdots (7)_{10}$

- Zweierkomplement:

  $-(8)_{10} \cdots (7)_{10}$

Darstellungsbereiche

Sei $x = (x_{n-1} \dots x_0)_2 \in B^n$ eine n-stellige Binärzahl.

1. Im Einerkomplement: $-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$
2. Im Zweierkomplement: $-2^{n-1} \leq x \leq 2^{n-1} - 1$

- Wegen der doppelten Darstellung der 0 im Einerkomplement lässt sich im Zweierkomplement eine Zahl mehr darstellen
- Einfache (technische) Umsetzung arithmetischer Operationen
- Addition bspw. genauso wie vorzeichenlos

Alternative Darstellung ganzer Zahlen

Bitfolge	Darstellung in Dezimalnotation		
	Vorz./Betrag	K_1	K_2
0000	+0	**+0**	+0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	**-0**	-1

BCD-Code

Binary **C**oded **D**ecimal
Ziffern werden einzeln mit 4 Bits als vorzeichenlose Binärzahl dargestellt.

Beispiel: 4739 = 0100 0111 0011 1001

6 Bitmuster ungenutzt, erlaubt Darstellung der Vorzeichen:
z.B. + = 1010 und - = 1011

Nachteil: Erschwerte Addition

Beispiel: 4739 + 1287 = 6026

0100	0111	0011	1001
0001	0010	1000	0111
0101	**1001**	**1100**	**0000**

Z.B. Fehler an der letzten Stelle wegen des Übertrags.

BCD-Code

Korrektur: Bei jedem Übertrag und bei jeder ungültigen BCD-Darstellung 6 aufaddieren:

0101	1001	1100	0000	→	Fehlerhaftes Ergebnis der letzten Rechnung
			0110		
0101	1001	1100	0110		
		0110		→	Korrektur
0101	1010	0010	0110		
	0110			→	Korrektur
0110	0000	0010	0110		
6	0	2	6		

Exkurs: Anwendungen BCD Code

- Vorteil von BCD: Einfache Konvertierung in Dezimalsystem, daher BCD Code für Anzeigen sinnvoll
- Konkrete Anwendungsbeispiele
  - DCF Funkuhr Signal
  - Bis heute existieren Hardwarebausteine (IC), welche u.a. BCD Codes für 7 Segment Displays dekodieren
  - Taschenrechner (u.a. von Texas Instruments) nutzen die freien Kombinationen für weitere Symbole (z.B. unendlich)

Abschnitt 1.3

Festkommazahlen

- Festkomma-Darstellung

Motivation: Festkommazahlen

- Fest vorgegebene Anzahl an Vor- und Nachkommastellen
- Getrennt voneinander binär dargestellt
- Häufig aus Performancegründen eingesetzt

Festkomma-Darstellung

- Komma **rechts** von der Stelle mit dem niedrigsten Wert:
  ein n-Bit Wort $(z_{n-1} \dots z_0)_2$ stellt dann die Zahl

$$z = \sum_{i=0}^{n-1} z_i \cdot 2^i \quad \text{dar,} \quad \text{z.B.} \quad 110101.0$$

- Komma **links** von der Stelle mit dem höchsten Wert:
  ein n-Bit Wort $(x_1 \dots x_n)_2$ stellt dann die Zahl

$$z = \sum_{i=1}^{n} z_i \cdot 2^{-i} \quad \text{dar,} \quad \text{z.B.} \quad 0.110101$$

- Allgemein stellt eine Bitfolge $(x_{n-1}, \dots, x_1, x_0, x_{-1}, \dots, x_{-m})_2$,
  falls das Komma zwischen x_0 und x_{-1} angenommen wird, die Zahl

$$z = \sum_{i=-m}^{n-1} z_i \cdot 2^i \quad \text{dar,} \quad \text{z.B.} \quad 1101.01$$

Beispiel für Festkomma-Arithmethik

- Verwende 4 Bits für Vor- und 3 Bits für Nachkommateil

$$
\begin{aligned}
3.5 + 2.5 &= (0011).(100) + (0010).(100) \\
&= (0110).(000) \\
&= 6.0
\end{aligned}
$$

Beispiel

$0{,}6875 \cdot 2 = 1{,}375$

$0{,}375 \cdot 2 = 0{,}75$

$0{,}75 \cdot 2 = 1{,}5$

$0{,}5 \cdot 2 = 1{,}0$

$\underline{0}$

$$(0{,}6875)_{10} = 0{,}(1011)_2$$

Abschnitt 1.4

Gleitkommazahlen

- Gleitkomma-Darstellung
- Regeln für das Rechnen mit Gleitkommazahlen
- Rechnerinterne Darstellung von Gleitkommazahlen
- IEEE 754

Motivation: Gleitkommazahlen

- Alternative zu Festkomma: approximative Darstellung reeller bzw. rationaler Zahlen
- Bei gleicher Anzahl an Bits wie bei Festkommadarstellung wird viel größerer Zahlenbereich abgedeckt
- Aber: Nicht jede Zahl in diesem Bereich kann exakt dargestellt werden
- Bis in die 80er Jahre hinein gab es viele verschiedene Gleitkomma-Darstellungen.
- Das Institute of Electrical and Electronics Engineers (IEEE) gab einen Standard für 32-, 64- und 80-Bit Gleitkommazahlen heraus.
- Probleme durch die Behandlung von Über- bzw. Unterläufen und anderen Ausnahmen.

Gleitkomma-Darstellung

- Jede Zahl z wird in der Form $z = \pm m \cdot b^{\pm e}$ dargestellt mit m **Mantisse**, e **Exponent**, b **Basis** für den Exponenten.

- Die Basis ist für alle auftretenden Exponenten die gleiche; daher rechnerinterne Darstellung einer Gleitkomma-Zahl:

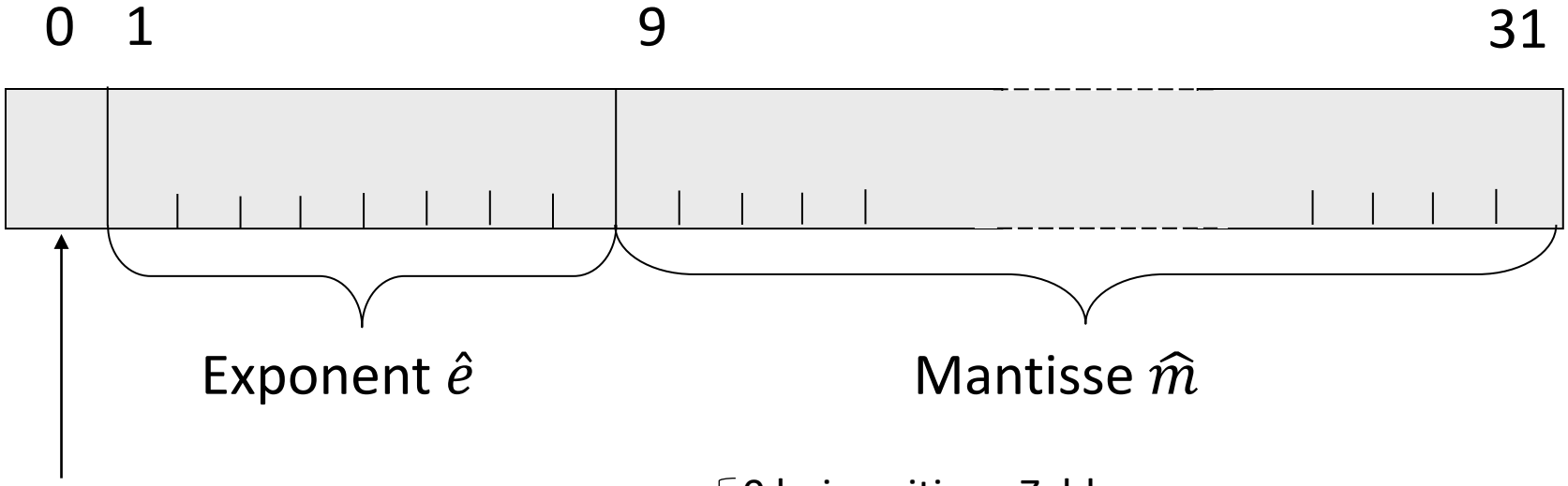
$$(\pm m, \pm e)$$

- Beispiel: $e = 1.6 * 10^{-19} C$

Bit-weise Darstellung

- Speicherung einer Gleitkomma-Zahl in drei Feldern
  - Vorzeichen $\hat{s}$
  - Exponent $\hat{e} = (e_{k-1} \dots e_0)$
  - Mantisse $\hat{m} = (m_{n-1} \dots m_0)$
- Bei 32-Bit haben wir $k = 8$ und $n = 23$
- Bei 64-Bit haben wir $k = 11$ und $n = 52$
- Drei Fälle: normalisiert, denormalisiert und Sonderzahlen

IEEE 754 32-bit („single“)



- Mit dieser Aufteilung darstellbare Beträge:

$$1.18 \cdot 10^{-38} \text{ bis } 3.40 \cdot 10^{38}$$

Normalisierte Darstellung

- $\hat{e} \neq (0 \ldots 0)$ und $\hat{e} \neq (1 \ldots 1)$
- Exponent $e = \hat{e} - bias$ mit $bias = 2^{k-1} - 1$
  - Für 32-Bit also $bias = 127$ da $k = 8$
- m wird interpretiert als $m = 1 + (0, \hat{m}) = 1, \hat{m}$
  - Darstellungstrick für ein weiteres Bit an Präzision
- Also gilt $1 \leq m < 2$

Normalisierte Darstellung

- Zum Beispiel wird die Zahl $z = -1.5$ dargestellt durch

$$10111111111000000000000000000000$$

d.h.

$\hat{s} = 1$ $\quad\quad$ $s = 1$

$\hat{e} = 01111111$ $\quad\quad$ $e = \hat{e} - 127 = 0$

$\hat{m} = 10000000000000000000000$ $\quad\quad$ $m = 1.\hat{m} = 1.5$

Normalisierte Darstellung

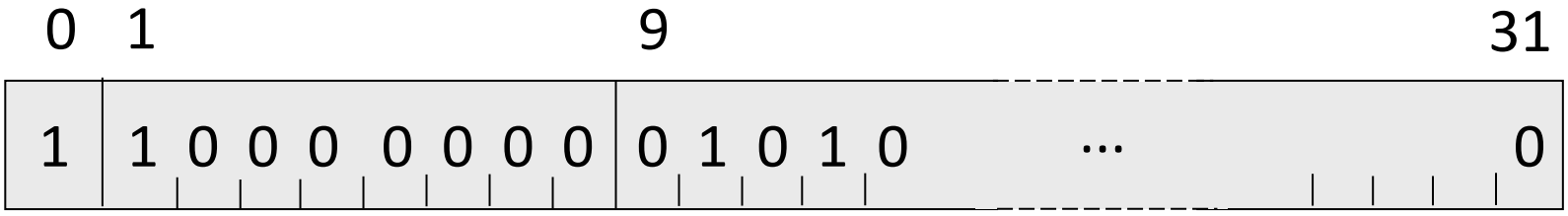
- Stelle -2.625 als IEEE 754 Gleitkommazahl dar

1. Vorzeichen: $s = 1 \rightarrow \hat{s} = 1$

2. Mantisse: $(2{,}625)_{10} = (10{,}101)_2$

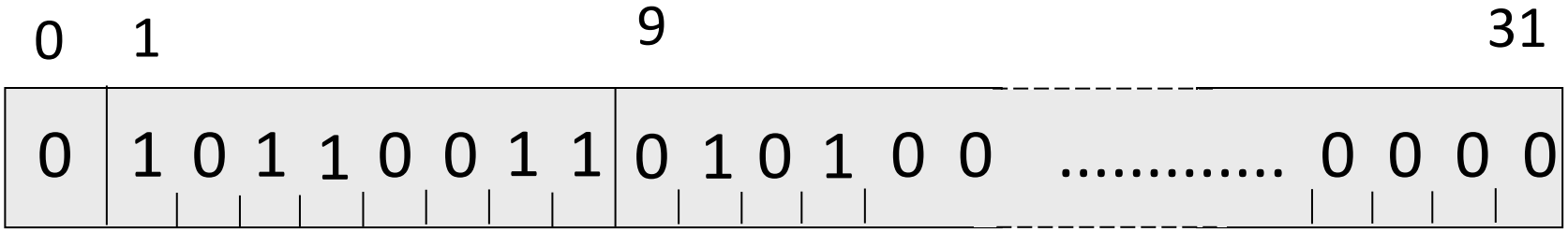
   Normalisieren: $(10{,}101)_2 = (\underline{\mathbf{1}}{,}0101)_2 \cdot (2)^1$
   $\hat{m} = 01010\ldots$

3. Exponent: $e = \hat{e} - 127 \leftrightarrow \hat{e} = e + 127$
   $\hat{e} = 1 + 127 = 128 = (10000000)_2$



Normalisierte Darstellung

- Welcher Dezimalzahl entspricht die folgende Binärzahl?



1. Vorzeichen: $\hat{s} = 0 \rightarrow s = 0 \rightarrow$ pos. Zahl
2. Exponenten: $\hat{e} = (10110011)_2 = (179)_{10}$
   $e = \hat{e} - 127 = 52$
3. Mantisse: $m = 1{,}\hat{m} = (1{,}0101)_2 = (1{,}3125)_{10}$

Ergebnis: $z = + 1{,}3125 \cdot 2^{52}$

Denormalisierte Darstellung

- Normalisiert lässt sich die 0 nicht darstellen wegen $1 \leq m$
- Falls $\hat{e} = (0 \ldots 0)$, dann denormalisiert
  - Exponent $e = 1 - bias$
  - Mantisse $m = \hat{m}$ ohne führende 1 $z = 0, \hat{m}$
- Genaue Darstellung für 0 und Werte nahe der 0
- Durch $1 - bias$ statt $0 - bias$ sanfter Übergang zwischen denormalisiert und normalisiert

Denormalisierte Darstellung

- Durch $1 - bias$ statt $0 - bias$ sanfter Übergang zwischen denormalisiert und normalisiert
- Kleinste normalisierte Zahl (Betrag)

$$\begin{aligned} \hat{m} &= (0 \ldots 0) \\ \hat{e} &= (0 \ldots 1) \end{aligned} \quad \Rightarrow \quad 1.0 \cdot 2^{-126}$$

- Größte denormalisierte Zahl (Betrag)

$$\begin{aligned} \hat{m} &= (1 \ldots 1) \\ \hat{e} &= (0 \ldots 0) \end{aligned} \quad \Rightarrow \quad \begin{aligned} &(0.11111111111111111111111)_2 \cdot 2^{-126} \\ &= (0.99999988079071044921875)_{10} \cdot 2^{-126} \end{aligned}$$

Darstellung der Null

- Kurioserweise gibt es zwei Darstellungen für die Zahl 0:

$$\left.\begin{array}{c}\hat{s} = 0 \\ \hat{m} = (0 \ldots 0) \\ \hat{e} = (0 \ldots 0)\end{array}\right\} \Rightarrow +0 \qquad \left.\begin{array}{c}\hat{s} = 1 \\ \hat{m} = (0 \ldots 0) \\ \hat{e} = (0 \ldots 0)\end{array}\right\} \Rightarrow -0$$

- Nach IEEE 754 manchmal gleich interpretiert (bei Vergleich z.B.), manchmal unterschiedlich

Sonderfall: Über- und Unterlauf

- **Überlauf:** Das Resultat einer Gleitkomma-Operation ist zu groß, um es darzustellen (bzw. zu klein, dann **Unterlauf**)

- Das Ergebnis einer solchen Gleitkomma-Operation wird durch **„infinity“** gekennzeichnet, Symbole $+\infty$ oder $-\infty$

- Beispiel:

$$\frac{1}{0} = +\infty \qquad \frac{1}{-0} = -\infty$$

- Im IEEE 754 wird dies durch $\hat{m} = (0 \ldots 0)$ und einen Exponenten $\hat{e} = (1 \ldots 1)$ mit $\hat{s} \in \{0,1\}$ dargestellt

Ausnahmebedingungen: NaN

- Wenn das Resultat einer Gleitkomma-Operation keine gültige Gleitkommazahl ist, dann wird eine Sonderzahl mit der Bedeutung **„not a number“ (NaN)** generiert.
  - Beispiel:

$$+\infty + (-\infty) = NaN$$

- *NaN* werden durch $\hat{m} \neq (0 \ldots 0)$ und einen Exponenten $\hat{e} = (1 \ldots 1)$ dargestellt

Approximative Darstellung

- Es können nicht alle Zahlen dargestellt werden
- Beispiel: $(0.1)_{10}$

$\hat{s} = 0$

$\hat{e} = (01111011)$

$\hat{m} = (10011001100110011001101)$

$(1.10011001100110011001101)_2 \cdot 2^{-4}$
$= 0.100000001490116119384765625$

- Nächstkleinere Zahl

$\hat{s} = 0$

$\hat{e} = (01111011)$

$\hat{m} = (10011001100110011001100)$

$(1.10011001100110011001100)_2 \cdot 2^{-4}$
$= 0.099999994039535522460937 5$

Approximative Darstellung

- float a = 0.1; float b = 0.3;
- if (a == 0.1) ergibt true
- Aber if (a+b == 0.4) ergibt false

- Daher Vergleich mit Epsilon-Umgebung nötig

Rechnen mit Gleitkommazahlen

- Seien $x = m_x \cdot 2^{d_x}$

  $y = m_y \cdot 2^{d_y}$

- Falls $d_x \leq d_y$ dann

$$x + y = \left(m_x \cdot 2^{d_x - d_y} + m_y\right) \cdot 2^{d_y}$$
$$x - y = \left(m_x \cdot 2^{d_x - d_y} - m_y\right) \cdot 2^{d_y}$$

Rechnen mit Gleitkommazahlen

- Seien $x = m_x \cdot 2^{d_x}$

  $y = m_y \cdot 2^{d_y}$

- Falls $d_x \leq d_y$ dann

$$x + y = \left(m_x \cdot 2^{d_x - d_y} + m_y\right) \cdot 2^{d_y}$$
$$x - y = \left(m_x \cdot 2^{d_x - d_y} - m_y\right) \cdot 2^{d_y}$$

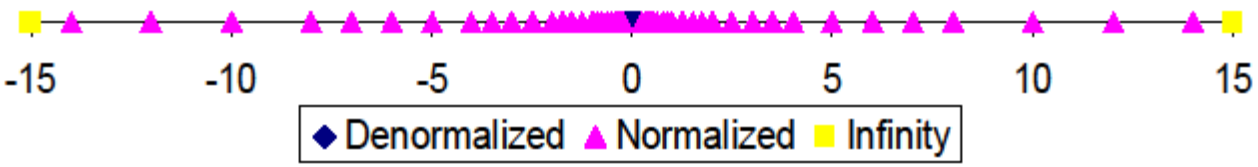
- Ebenso $x \cdot y = \left(m_x \cdot m_y\right) \cdot 2^{d_x + d_y}$

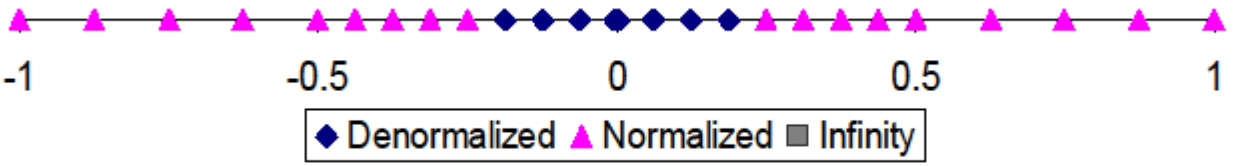
$$\frac{x}{y} = \frac{m_x}{m_y} \cdot 2^{d_x - d_y}$$

Verteilung von Gleitkommazahlen

- Vorzeichen: 1 Bit
- Exponent: 3 Bits
- Mantisse: 2 Bits

- Weicher Übergang zwischen normalisierten und denormalisierten Werten
- Präzise Verteilung um die 0





Computer Systems: A Programmer's Perspective (Bryant, O'Hallaron)

IEEE 754 - 64-Bit Zahlen

- Genau wie bei den vorgestellten 32-Bit Gleitkomma-Zahlen, jedoch:
  - Der Exponent besitzt 11 Bits (vorher: 8 Bits).
  - Die Mantisse ist 52 Bits lang (vorher: 23 Bits).

- Außerdem beschreibt IEEE 754 auch 80-Bit Gleitkomma-Zahlen mit 15-Bit Exponenten und 64-Bit Mantisse

- x86-Architekturen verwenden interne 80-Bit Darstellung, Speicherung erfolgt in 32- oder 64-Bit
- 64-Bit Gleitkomma-Zahlen werden auch als **double** bezeichnet.

Achtung: Rechnen mit Gleitkommazahlen

- Mit $x, y \in R$ ist das Ergebnis einer IEEE 754 Operation
  $\circ \in \{+, -, \cdot, /\}$ definiert als $round(x \circ y)$

- IEEE 754 definiert nicht, wie exakt gerundet werden muss, sondern liefert Alternativen
  - Round to zero
  - Round down
  - Round up
  - Round to nearest
    - Tie to even
    - Tie away from zero

- D.h. die gleichen 32-Bit FP-Operationen können unterschiedliche Ergebnisse liefern, obwohl korrekt (den Standard befolgend)

Beispiel: Rundung

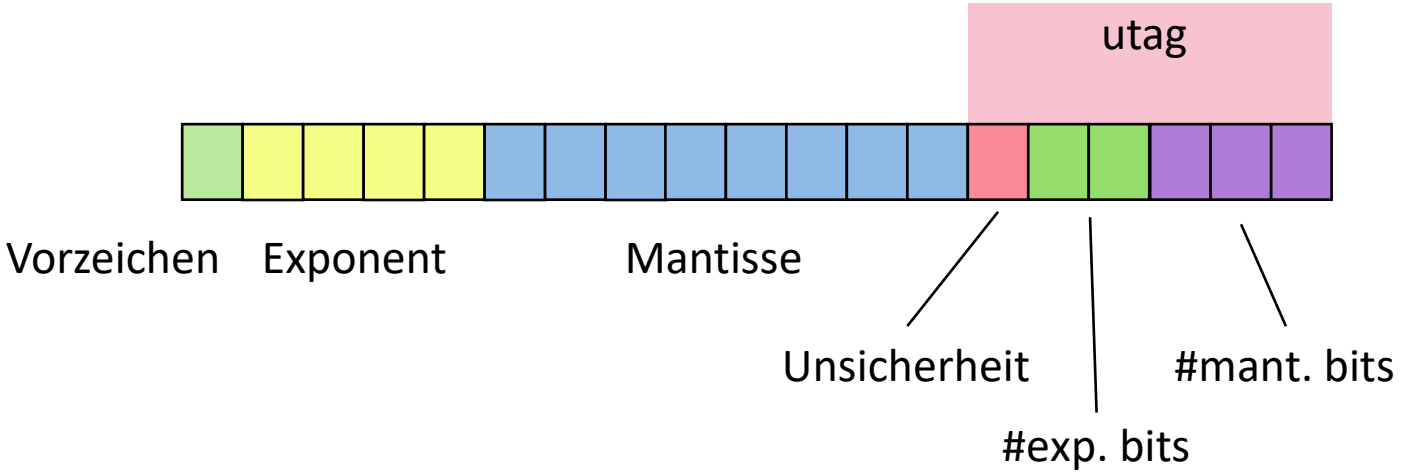
Modus	1.40	1.60	1.50	2.50	-1.50
Round to nearest (tie to even)	1	2	2	2	-2
Round to nearest (tie away from zero)	1	2	2	3	-2
Round toward zero	1	1	1	2	-1
Round down	1	1	1	2	-2
Round up	2	2	2	3	-1

Alternative Darstellung: UNUMs

- Motivation:
  - Gleitkommazahlen bilden keine bekannte algebraische Struktur
  - Rundungsfehler sind sicherheitskritisch
  - Keine einfache parallele Berechnung von Operationen auf Gleitkommazahlen möglich

Universal Numbers

- Struktur (Größe variabel)



- Unsicherheit:
  - 0: exakt dargestellte Zahl
  - 1: Intervall zwischen zwei exakt darstellbaren zahlen
- #exp.bits: genutzter Bereich des Exponenten
- #mant.bits: genutzter Bereich der Mantisse

Universal Numbers

Vorteile:

- Keine Rundungen, kein Genauigkeitsverlust
- Unterliegen bekannten algebraischen Gesetzen
- Berechnung parallelisierbar
- Kompakter als IEEE Gleitkommazahlen

Nachteile:

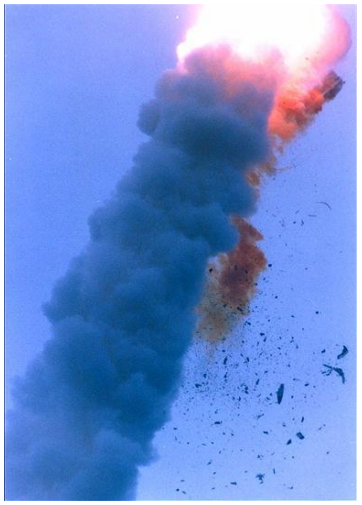
- Mehr Prozessorlogik erforderlich
- Nicht etabliert

Abschnitt 1.5

Zusammenfassung

Achtung: Fehler kosten (viel) Geld

- Ariane 5 stürzte 37 Sekunden nach dem Start ab
- Kosten in Höhe von mehreren Hundert Millionen US-Dollar
- Was ist passiert?
- Konvertierung eines 64-Bit Floats in einen 16-Bit Signed Integer hat zu einem Überlauf geführt



Quelle: http://www.capcomespace.net/dossiers/espace_europeen/ariane/ariane5/AR501/V88_AR501.htm

Zusammenfassung

- Darstellung von natürlichen und ganzen Zahlen sowie deren wichtigsten Rechenoperationen
- BCD-Kodierung
- Festkommadarstellung und IEEE 754 Fließkommadarstellung
- Wichtig: Operationen verhalten sich nicht immer wie erwartet

Literatur

- W. Kahan: An Interview with the Old Man of Floating-Point (https://people.eecs.berkeley.edu/~wkahan/ieee754status/754story.html)

- D. Goldberg: *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. In: *ACM Computing Surveys*. 23, 1991, S. 5-48 (http://docs.sun.com/source/806-3568/ncg_goldberg.html)

- R. Bryant, D. O'Hallaron: Computer Systems – A Programmer's Perspective (Chapter 2). Prentice Hall