

IT-Infrastrukturen – Rechnerstrukturen

Thema 1: Zahlendarstellung und Rechnerarithmetik

Prof. Dr.-Ing. Sebastian Schlesinger
Professur für Wirtschaftsinformatik
(Infrastruktur & Security)

Dieser Foliensatz basiert auf den Folien
zu „Rechnerorganisation“ von Prof. Dr.
Ben Juurlink (TU Berlin) und Prof. Dr.
Paula Herber, WWU Münster



- Nach dieser Vorlesung sollten Sie in der Lage sein:
 - **Binär-/Oktal-/Hexadezimalzahlen** zu Dezimalzahlen zu konvertieren und umgekehrt
 - **2-Komplement-Zahlen** zu berechnen und zu negieren
 - m-Bit 2-Komplement-Zahlen zu n-Bit zu konvertieren
 - **Arithmetische Operationen mit Binärzahlen** durchzuführen
 - einen **Überlauf (overflow)** zu erklären und zu erläutern, wann er auftritt und wie er behandelt wird
 - eine rationale Zahl in eine **Gleitkommazahl** (IEEE 754 Floating Point Standard) zu konvertieren und umgekehrt
 - **Arithmetische Operationen mit Gleitkommazahlen** durchzuführen

Übersicht über die heutige Vorlesung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“

Übersicht über die heutige Vorlesung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“



Natürliche Zahlen können in jeder Basis repräsentiert werden:

$$a_{n-1}a_{n-2}\dots a_1a_0(\text{Basis } B) = a_{n-1}B^{n-1} + \dots + a_1B^1 + a_0B^0 = \sum_{i=0}^{n-1} a_i B^i$$

- B: **Basis**, z.B. 10 (dezimal), 2 (binär), 8 (oktal), 16 (hexadezimal)
- B^i : **Gewicht der i-ten Ziffer**
- a_i : i-te Ziffer aus der Menge $\{0, 1, \dots, B-1\}$

Beispiel (dezimal): 2435 = $2 \cdot 10^3 + 4 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$

Diagram illustrating the decomposition of the decimal number 2435 into its positional weights and digits:

- $B^3 = 1000$ (weight for a_3)
- $B^2 = 100$ (weight for a_2)
- $B^1 = 10$ (weight for a_1)
- $B^0 = 1$ (weight for a_0)

The digits a_3, a_2, a_1, a_0 are shown above and below the respective terms in the expansion.



Natürliche Zahlen können in jeder Basis repräsentiert werden:

$$a_{n-1}a_{n-2} \dots a_1a_0(\text{Basis } B) = a_{n-1}B^{n-1} + \dots + a_1B^1 + a_0B^0 = \sum_{i=0}^{n-1} a_i B^i$$

- B: **Basis**, z.B. 10 (dezimal), 2 (binär), 8 (oktal), 16 (hexadezimal)
- B^i : **Gewicht der i-ten Ziffer**
- a_i : i-te Ziffer aus der Menge $\{0, 1, \dots, B-1\}$

Beispiel (binär):

$$\begin{array}{ccccccc} & a_3 & a_2 & a_1 & a_0 & & \\ & \downarrow & \downarrow & \downarrow & \downarrow & & \\ 1 & 0 & 1 & 1 & & & \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & & \\ a_3 & a_2 & a_1 & a_0 & & & \end{array} \quad \begin{array}{ccccccc} B^3=8 & B^2=4 & B^1=2 & B^0=1 & & & \\ \downarrow & \downarrow & \downarrow & \downarrow & & & \\ = & 1 \cdot 2^3 & + 0 \cdot 2^2 & + 1 \cdot 2^1 & + 1 \cdot 2^0 & = & (8 + 0 + 2 + 1)_{10} = 11_{10} \end{array}$$



- Menschen benutzen **Dezimalzahlen**

$$2435 = 2 \cdot 10^3 + 4 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

- Rechner benutzen **Binärzahlen**

$$1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

- Hintergrund: Zwei Möglichkeiten einfach darstellbar

- ein / aus
- hohes / niedriges Potenzial

- Interpretation von Binärzahlen:

$$b_{n-1}b_{n-2} \dots b_1b_0 = b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Konvertierung



- Zerlegung nach **Horner-Schema**:

$$\sum_{i=0}^{n-1} a_i B^i = (((... (a_{n-1} B + a_{n-2}) B + ... + a_2) B + a_1) B + a_0$$

- Dezimal nach dual / binär:

$167_D \rightarrow 167 / 2 = 83$	Rest 1
$83 / 2 = 41$	Rest 1
$41 / 2 = 20$	Rest 1
$20 / 2 = 10$	Rest 0
$10 / 2 = 5$	Rest 0
$5 / 2 = 2$	Rest 1
$2 / 2 = 1$	Rest 0
$1 / 2 = 0$	Rest 1

Niederwertigstes Bit
(least significant bit (LSB))

Höchstwertigstes Bit
(most significant bit (MSB))

Rechnen Sie 202_D in eine Binärzahl um!

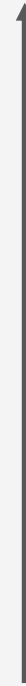
Wie viele Dezimalzahlen lassen sich mit einer 3-stelligen Binärzahl darstellen?

- $167_D = 10100111_B$

Konvertierung: Beispiel



$202_D \rightarrow$	$202 / 2 = 101$	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



■ $202_D = 11001010_B$



- Alle Informationen werden im Rechner als Binärzahlen gespeichert
- kleinste Einheit: 1 Bit = eine Ziffer einer Dualzahl
- Beispiel:

höchstwertigstes (*most significant*) Bit

1Byte = 8bit = 1100 1010

niederwertigstes (*least significant*) Bit

- Mit einer n-Bit Zahl lassen sich die Werte von $0 \dots 2^n - 1$ darstellen
- Beispiel für $n=3$:

$$111_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 7 = 8 - 1 = 2^3 - 1$$



- **Problem:** Binärzahlen werden schnell sehr lang
- **Lösung:** Verwendung von weiteren Zahlensystemen mit höheren Basen, in die leicht konvertiert werden kann:
 - Oktale Zahlen (Basis 8)
 - Hexadezimale Zahlen (Basis 16)

Konvertierung nach Oktal



- Oktalzahlen werden zur **Basis 8** ($= 2^3$) gebildet
- Dezimal nach oktal (**Horner-Schema**): $167_D \rightarrow$

$167 / 8 = 20$	Rest 7
$20 / 8 = 2$	Rest 4
$2 / 8 = 0$	Rest 2

 $167_D = 247_O$

- Binär nach oktal: es werden 3 Binärziffern benötigt, um die Zahlen 0 ... 7 darzustellen

$$O_0 * 8^0 = b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$$

- jeweils **3 Binärziffern** können als eine Oktalziffer zusammengefasst werden

$$\begin{array}{ccccccc} & & \mathbf{0} & & \mathbf{3} & & \\ & & \underbrace{} & & \underbrace{} & & \\ \mathbf{111000001011}_2 = & \underbrace{111}_7 & 000 & 001 & \underbrace{011}_1 & _2 = & 7013_8 \end{array}$$

Hexadezimalzahlen



- Hexadezimalzahlen werden zur **Basis 16** ($= 2^4$) gebildet
- Ziffernmenge: $\{0, 1, 2, \dots, 8, 9, A, B, C, D, E, F\}$
- Dezimal nach hexadezimal: $167_D \rightarrow 167 / 16 = 10 \text{ Rest } 7$
 $10 / 16 = 0 \quad \text{Rest A (entspricht 10)}$
 $167_D = A7_H$

In C/Java: vorgestelltes „0x“ kennzeichnet Hexadezimalzahlen (0xa7)

- Binär nach hexadezimal: es werden 4 Binärziffern benötigt, um die Zahlen 0 ... 15 darzustellen:
$$h_0 * 16^0 = b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$$
- jeweils **4 Binärziffern können als eine Hexadezimalziffer zusammengefasst** werden
- Binär nach hexadezimal:

$$\begin{array}{ccccccccc} & & \mathbf{3} & & \mathbf{7} & & \mathbf{B} & & \mathbf{F} \\ & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \\ 0001 & 0011 & 0101 & 0111 & 1001 & 1011 & 1101 & 1111 & \\ \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \\ \mathbf{1} & & \mathbf{5} & & \mathbf{9} & & \mathbf{D} & & \end{array} \quad {}_2 = 13579BDF_{16}$$

Übersicht über die heutige Vorlesung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“

Negative Zahlen



Möglichkeiten negative Zahlen binär zu repräsentieren:

- **Vorzeichen-/Betrags-Zahlen** (*Sign-magnitude numbers*)
 - MSB zeigt Vorzeichen (sign) an (0: positiv, 1: negativ).
 - Die übrigbleibenden ($n - 1$) Bits bilden den Betrag (magnitude).
 - Beispiel: 5 (dezimal) = 0101 (binär) \rightarrow - 5 (dezimal) = 1101 (binär)
- **1-Komplement-Zahlen** (*One's complement numbers*)
 - Zahl wird durch die Invertierung aller Bits negiert.
 - MSB impliziert das Vorzeichen.
 - Beispiel: 5 (dezimal) = 0101 (binär) \rightarrow - 5 (dezimal) = 1010 (binär)
- **2-Komplement-Zahlen** (*Two's complement numbers*)
 - MSB hat ein negatives Gewicht (-2^{n-1}).
 - $b_{n-1}b_{n-2}...b_1b_0$ (binär) = $-b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + ... + b_12^1 + b_02^0$ (dezimal)
 - MSB impliziert das Vorzeichen.
 - Beispiel: - 5 (dezimal) = $-8 + 3 = 1011$ (binär)

Beispiel (3 Bit)



Dezimal	Vorzeichen- / Betrags-Zahlen (Sign-Magnitude)	1-Komplement- Zahlen (One's complement)	2-Komplement- Zahlen (Two's complement)
-4			100
-3	111	100	101
-2	110	101	110
-1	101	110	111
-0	100	111	
+0	000	000	000
+1	001	001	001
+2	010	010	010
+3	011	011	011

Welche Darstellung würden Sie wählen?

Was sind die kleinste und größte mit 8 Bit repräsentierbare Ganzzahl?

Und der Gewinner ist ...



■ 2-Komplement-Zahlen

- Arithmetik ist einfacher (identisch zu vorzeichenlos !).
- Es gibt nur eine Möglichkeit die 0 zu repräsentieren.

■ Beispiel 32bit:

- $b_{31}b_{30}...b_1b_0$ (binär) = $-b_{31}2^{31}+b_{30}2^{30}+...+b_12^1+b_02^0$ (dezimal)
- kleinste mit 8 Bit darstellbare Zahl: $-2^7 = -128$ (-2^{n-1})
- größte mit 8 Bit darstellbare Zahl: $2^6 + 2^5 + ... + 2^0 = 127$ ($= 2^{n-1} - 1$)

Negation



- Negation von 2-Komplement-Zahlen:
 - Invertiere alle Bits und addiere 1
- 8-Bit-Beispiel: $0110\ 1001_B = +105_D$

invertieren: $1001\ 0110$

1 addieren: $1001\ 0111 = -128 + 16 + 7 = -105$

Rückwärts:

invertieren: $0110\ 1000$

1 addieren: $0110\ 1001$

Übersicht über die heutige Vorlesung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“

Binäre Addition



- Addition von rechts nach links mit Übertrag (*carry*) wie in der Grundschule
- Beispiele (4-Bit):

$$\begin{array}{r} \text{carry} \rightarrow 1 \\ 0010 \\ + 0011 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 2 \\ + 3 \\ \hline 5 \end{array}$$

Rechenregeln Addition

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$
carry ↑

Binäre Subtraktion



■ Beispiele (4-Bit):

$$\begin{array}{r} 0110 \\ - 0001 \\ \hline 0101 \end{array}$$

borrows (indicated by a blue arrow pointing from the result '1' to the first '0' in the minuend)

$$\begin{array}{r} 6 \\ - 1 \\ \hline 5 \end{array}$$

Rechenregeln:

- $0 - 0 = 0$
- $0 - 1 = 11$ (*borrow*)
- $1 - 0 = 1$
- $1 - 1 = 0$

Berechnen Sie jeweils $5 + 6$ und $-3 - 6$!
Sind die Ergebnisse korrekt?

Binäre Addition



- Addition von rechts nach links mit Übertrag (*carry*) wie in der Grundschule
- Beispiele (4-Bit):

$$\begin{array}{r} \text{carry} \rightarrow 1 \\ 0010 \\ + 0011 \\ \hline 0101 \\ \text{carry} \rightarrow 1 \\ 0101 \\ + 0110 \\ \hline 1011 \end{array}$$

$$\begin{array}{r} 2 \\ + 3 \\ \hline 5 \\ 5 \\ + 6 \\ \hline - 5 \end{array}$$

Rechenregeln:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$

carry ↗

⇒ Überlauf (*overflow*)

Binäre Subtraktion



■ Beispiele (4-Bit):

	0110	6
	- 0001	- 1
	<hr/>	
	1	
borrows	0101	5
	1101	- 3
	- 0110	- 6
	<hr/>	
	11	
	0111	7

⇒ Überlauf (*overflow*)

Rechenregeln:

- $0 - 0 = 0$
- $0 - 1 = 11$ (*borrow*)
- $1 - 0 = 1$
- $1 - 1 = 0$

■ Überläufe müssen erkannt und angezeigt werden!

Subtraktion durch Addition der Negation



- Beispiel (4-Bit):

$$\begin{array}{r} 0010 \\ - 0011 \\ \hline \end{array} \qquad \begin{array}{r} 2 \\ - 3 \\ \hline \end{array}$$

- Negiere 0011

- Invertieren: 1100
- 1 addieren: 1101

- Addieren

$$\begin{array}{r} 0010 \\ + 1101 \\ \hline 1111 \end{array} \qquad \begin{array}{r} 2 \\ - 3 \\ \hline -1 \end{array}$$



- **Überlauf (overflow):**

- Das Ergebnis ist zu groß um mit der gegebenen Anzahl an Bits gespeichert zu werden.
- z.B. Addition von zwei n-Bit-Zahlen muss keine n-Bit-Zahl ergeben.

- Kein Überlauf, wenn ...

- Addition von 2 Zahlen mit entgegengesetztem Vorzeichen
 - Beispiel: $-10 + 6 = -4$
- Subtraktion von 2 Zahlen mit demselben Vorzeichen

- Überlauf tritt auf, wenn...

Operation	A	B	Ergebnis
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

**An welchen Bits
können wir das
erkennen?**

- Ein Überlauf tritt genau dann auf, wenn das Übertragsbit (*carry in*) für das MSB ungleich dem entstehenden Übertragsbit (*carry out*) aus der Operation ist.

carry in \neq *carry out* \rightarrow Überlauf

- Beispiel (4-Bit):

carry out \longrightarrow 01 \longleftarrow carry in

$$\begin{array}{r} 0111 \\ + 0001 \\ \hline 1000 \end{array}$$

Operation	A	B	Ergebnis	carry out	carry in
A+B	≥ 0	≥ 0	< 0	0	1
A+B	< 0	< 0	≥ 0		
A-B	≥ 0	< 0	< 0		
A-B	< 0	≥ 0	≥ 0		

- Ein Überlauf tritt genau dann auf, wenn das Übertragsbit (*carry in*) für das MSB ungleich dem entstehenden Übertragsbit (*carry out*) aus der Operation ist.

carry in \neq *carry out* \rightarrow Überlauf

- Beispiel (4-Bit):

carry out \longrightarrow 10 \longleftarrow carry in

$$\begin{array}{r} 1000 \\ + 1111 \\ \hline 0111 \end{array}$$

Operation	A	B	Ergebnis	carry out	carry in
A+B	≥ 0	≥ 0	< 0	0	1
A+B	< 0	< 0	≥ 0	1	0
A-B	≥ 0	< 0	< 0		
A-B	< 0	≥ 0	≥ 0		

- Ein Überlauf tritt genau dann auf, wenn das Übertragsbit (*carry in*) für das MSB ungleich dem entstehenden Übertragsbit (*carry out*) aus der Operation ist.

carry in \neq *carry out* \rightarrow Überlauf

- Beispiel (4-Bit):

carry out \longrightarrow 10 \longleftarrow carry in

$$\begin{array}{r} 0111 \\ - 1111 \\ \hline 1000 \end{array}$$

Operation	A	B	Ergebnis	carry out	carry in
A+B	≥ 0	≥ 0	< 0	0	1
A+B	< 0	< 0	≥ 0	1	0
A-B	≥ 0	< 0	< 0	1	0
A-B	< 0	≥ 0	≥ 0		

- Ein Überlauf tritt genau dann auf, wenn das Übertragsbit (*carry in*) für das MSB ungleich dem entstehenden Übertragsbit (*carry out*) aus der Operation ist.

carry in \neq *carry out* \rightarrow Überlauf

- Beispiel (4-Bit):

carry out \longrightarrow 01 \longleftarrow carry in

$$\begin{array}{r} 1000 \\ - 0001 \\ \hline 0111 \end{array}$$

Operation	A	B	Ergebnis	carry out	carry in
A+B	≥ 0	≥ 0	< 0	0	1
A+B	< 0	< 0	≥ 0	1	0
A-B	≥ 0	< 0	< 0	1	0
A-B	< 0	≥ 0	≥ 0	0	1

Multiplikation



- Für den Moment betrachten wir nur positive Zahlen.
- Schulmathematik: Es wird immer die ganze linke Zahl mit einer Stelle der rechten Zahl multipliziert (oder umgekehrt).

$$\begin{array}{r} 1101 \cdot 1011 \\ \hline \end{array}$$

1101

1101

0000

1101

Produkt

10001111

$$\begin{array}{r} 13 \cdot 11 \\ \hline \end{array}$$

13

13

143

- Produkt kann doppelte Stellenanzahl erfordern!

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend		Divisor		Quotient
1001010	/	1000	=	

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend Divisor Quotient

1001010 / 1000 = 1

1000
1

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend		Divisor		Quotient
1001010	/	1000	=	10
<div>1000 — 10</div>				

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend		Divisor	=	Quotient
1001010	/	1000	=	100
<u>1000</u>				
101				

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend		Divisor	=	Quotient
1001010	/	1000	=	1001
<div>1000</div> <div>1010</div>				

A blue arrow points from the '1' in the second position of the dividend (1001010) to the '1' in the second position of the quotient (1001).

Division



- Schulmathematik: Es wird immer ein Teil der linken Zahl (Dividend) durch die gesamte rechte Zahl (Divisor) geteilt.

Dividend		Divisor	=	Quotient	
1001010	/	1000	=	1001	74/8= 9 Rest 2
1000					
1010				= 9	
1000					
10	→ Rest				
= 2					

Multiplikation und Division



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Berechnen Sie:

$101 \cdot 011$

$1110 / 0010$

Multiplikation und Division



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Berechnen Sie:

$$\begin{array}{r} 101 \cdot 011 \\ \hline 101 \\ 101 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} 10101 / 0111 = 11 \\ \underline{0111} \downarrow \\ 00111 \\ \underline{0111} \\ 0000 \end{array}$$

Übersicht über die heutige Vorlesung



Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“



- Darstellung rationaler Zahlen:

$$\begin{aligned} & a_{n-1}a_{n-2} \dots a_0, a_{-1}a_{-2} \dots a_{-m} \\ &= a_{n-1}B^{n-1} + \dots + a_0B^0 + a_{-1}B^{-1} + a_{-2}B^{-2} + \dots + a_{-m}B^{-m} \\ &= \sum_{i=0}^{n-1} a_i B^i + \sum_{i=-m}^{-1} a_i B^i \end{aligned}$$

- Beispiel (dezimal): $12,48_D = 2 \cdot 10^1 + 2 \cdot 10^0 + 4 \cdot 10^{-1} + 8 \cdot 10^{-2}$
- Beispiel (binär): $11,1010_B = 2^1 + 2^0 + 2^{-1} + 2^{-3}$
 $= 3 + 0,5 + 0,125 = 3,625_D$

Konvertierung


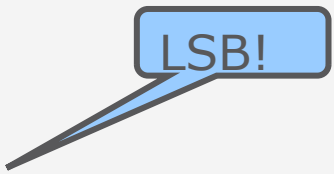


- Dezimal nach Dual

- $2 \cdot 0,b_{-1}b_{-2}\dots b_{-m} = b_{-1}b_{-2}b_{-3}\dots b_{-m}$

- Beispiel: $0,24_D$

$0,24_D \rightarrow$

$0,24 \cdot 2 = 0,48 + 0$	
$0,48 \cdot 2 = 0,96 + 0$	
$0,96 \cdot 2 = 0,92 + 1$	
$0,92 \cdot 2 = 0,84 + 1$	
$0,84 \cdot 2 = 0,68 + 1$	
$0,68 \cdot 2 = 0,36 + 1$	
$0,36 \cdot 2 = 0,72 + 0$	
$0,72 \cdot 2 = 0,44 + 1 \rightarrow 0,00111101_B$	

- Abbruch nach 8 Stellen (Näherung mit 0,238...)

Gleitkommazahlen (*floating point*)



- Näherung für reelle Zahlen:

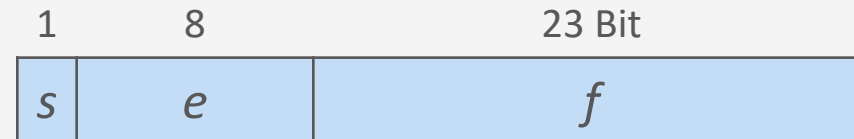
$$(-1)^s \cdot 1.f \cdot 2^E$$

- s : **Vorzeichen** (*sign*): 0 \rightarrow positiv, 1 \rightarrow negativ
- $1.f$: **Mantisse** (Betrag) als **normalisierte Zahl**
 - Zahl wird so lange geschoben, bis sie führende 1 aufweist
 - Binärpunkt wird rechts von dieser 1 festgelegt ($1.0 \leq 1.f < 2.0$)
- f : nur der **fraktionale Anteil** f (*fraction*) wird gespeichert, **führende 1 ist implizit** (wird von Recheneinheit ergänzt)
- E : vorzeichenbehafteter Exponent, wird als **transformierter Exponent** e gespeichert
- e : $e = E + \text{bias}$

IEEE-754 Standard

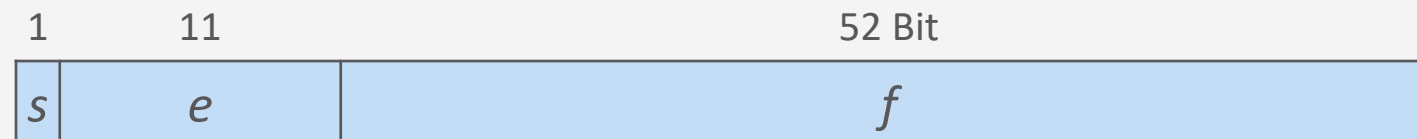


- Einfache Genauigkeit (*single precision*, 32 Bit)



- bias = 127
- C/Java: `float`

- Doppelte Genauigkeit (*double precision*, 64 Bit)



- bias = 1023
- C/Java: `double`

Beispiel



- $-0,75_D$ mit einfacher Genauigkeit

- $s = 1$

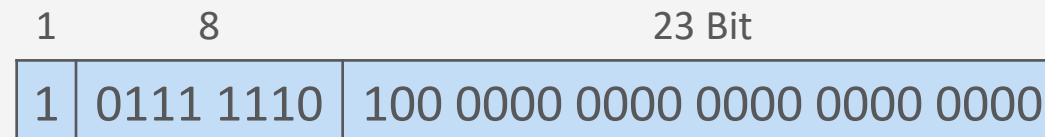
- $0,75_D$ als rationale Dualzahl ist $0,11_B$

- Normalisiere: $0,11 = 1,1 \cdot 2^{-1}$

- führende 1 ist implizit $\rightarrow f = 10000....$

- transformierter Exponent e

$$e = E + \text{bias} = -1 + 127 = 126 = 0111\ 1110_B$$



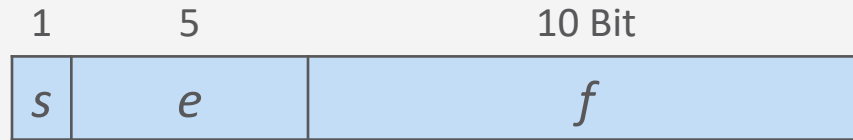


- Neben **normalisierten Zahlen** sind außerhalb des Zahlenraums definiert:
 - \pm Null
 - \pm Unendlich
 - \pm unnormalisierte (*unnormalized*) Zahlen: winzige (*tiny*) Zahlen
 - Nichtzahlen (Not a Number, NaN): Ergebnis ungültiger Operation wie 0/0
- Codiert durch den größten und kleinsten Exponentwert e und f :
 - normal.: $(-1)^s \cdot 1.f \cdot 2^{e-127}$ bzw. -1023 $1 \leq e \leq 254$ bzw. 2046
 - Null: $(-1)^s \cdot 0$ $e = 0, f = 0$
 - Unendlich: $(-1)^s \cdot \infty$ $e = 255$ bzw. 2047, $f = 0$
 - unnorm.: $(-1)^s \cdot 0.f \cdot 2^{-126}$ bzw. -1022 $e = 0, f \neq 0$; interpretiert mit $e = 1$
 - Nichtzahl: NaN $e = 255$ bzw. 2047, $f \neq 0$

Addition von Gleitpunktzahlen / 1



- Beispiel basiert auf **16-Bit Minifloat** Format:



bias = 15

Exponentenbereich: $-14 \leq E \leq 15$

- $Z = X + Y$ mit
 - $X = 2,35_D = 10.0101\ 1001\ 1001\ 1001 \dots_B$
 - $Y = 10,17_D = 1010.0010\ 1011\ 1000\ 0101 \dots_B$

Addition von Gleitpunktzahlen / 2



- $Z = X + Y$ mit
 - $X = 2,35_D = 10.0101\ 1001\ 1001\ 1001 \dots_B$
 - $Y = 10,17_D = 1010.0010\ 1011\ 1000\ 0101 \dots_B$
- **1. Schritt:**
 - Normalisieren und Anpassung an 16-Bit-Format:
 - $X = 1.0010\ 1100\ 11 \cdot 2^1$
 - $Y = 1.0100\ 0101\ 01 \cdot 2^3$
- **2. Schritt:** Vergleichen der beiden Exponenten E .
 - Bei Ungleichheit kleineren Exponent an den größeren anpassen
 - $X = 0.0100\ 1011\ 00\ 11 \cdot 2^3$

Wie hoch ist jeweils im 1. und 2. Schritt der Genauigkeitsverlust?

Addition von Gleitpunktzahlen / 3



- $Z = X + Y$ mit
 - $X = 2,35_D = 10.0101\ 1001\ 1001\ 1001 \dots_B$
 - $Y = 10,17_D = 1010.0010\ 1011\ 1000\ 0101 \dots_B$
- **1. Schritt:**
 - Normalisieren und Anpassung an 16-Bit-Format: Mantisse umfasst nur 10 Bit, Rest geht verloren
 - $X = 1.0010\ 1100\ 11 \cdot 2^1 = 2,349609375_D$ (Genauigkeitsverlust: 0,00039625)
 - $Y = 1.0100\ 0101\ 01 \cdot 2^3 = 10,1640625_D$ (Genauigkeitsverlust: 0,0059375)
- **2. Schritt:** Vergleichen der beiden Exponenten E .
 - Bei Ungleichheit kleineren Exponent an den größeren anpassen
 - $X = 0.0100\ 1011\ 00\ 11 \cdot 2^3$
 - rot dargestellten Stellen gehen verloren
 - $X = 0.0100\ 1011\ 00 \cdot 2^3 = 2,34375_D$ (Genauigkeitsverlust: 0,005859375)

Addition von Gleitpunktzahlen / 4



- **3. Schritt:** Addieren der Mantissen:

$$\begin{array}{r} 0.0100\ 1011\ 00\ (X) \\ +\ 1.0100\ 0101\ 01\ (Y) \\ \hline 1.1001\ 0000\ 01\ (Z) \end{array}$$

- **Ergebnis**

- muss ggf. noch normalisiert werden (hier nicht)
- $Z = 1.1001\ 0000\ 01 \cdot 2^3 = 12,5078125$ (korrekt wäre: 12,52, Genauigkeitsverlust: 0,0121875).

Übersicht über die heutige Vorlesung



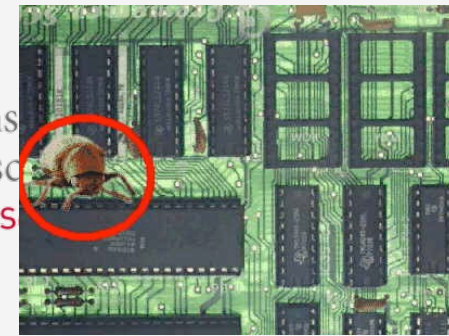
Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

- Zahlensysteme und ihre Konvertierung
- Vorzeichenbehaftete und vorzeichenlose Zahlen
- Arithmetische Operationen
 - Addition und Subtraktion
 - Multiplikation
 - Division
- Gleitkommazahlen (Floating Point)
- „Pentium Bug“

Pentium Bug



Hochs
Wirtsch
Berlin S



Fehler im Divisionsalgorithmus für Gleitpunktzahlen

- Juli 1994: Intel entdeckt Fehler. Geschätzte Kosten zum Beheben: **einige 100K\$**
- Sept. 1994: Matheprof Thomas Nicely entdeckt Fehler. Erhält keine offizielle Stellungnahme Intel, veröffentlicht Entdeckung im Internet
- 7. Nov. 1994: EE Times bringt Geschichte auf Titelseite
- 22. Nov. 1994: Pressemitteilung Intel: Pentium könne Fehler an 9. Stelle verursachen, nur wenige Benutzer könnten betroffen sein
- 5. Dez. 1994: Intel behauptet, Fehler würde nur einmal in 27000 Jahren auftreten bei typischer Anwendung
- 12. Dez. 1994: IBM Research ficht Intels Berechnung an: alle 24 Tage Fehler
- 21. Dez. 1994: Intel gibt zu: jeder Besitzer darf Pentium austauschen. Geschätzte Kosten: **500 M\$!**

$4195835.0/3145727.0 = 1.333\ 820\ 449\ 136\ 241\ 002$ (korrekter Wert)

$4195835.0/3145727.0 = 1.333\ 739\ 068\ 902\ 037\ 589$ (fehlerhafter Pentium)



- Computer benutzen **Binärzahlen** statt Dezimalzahlen
 - $b_{n-1}b_{n-2} \dots b_1b_0 = b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$
- Binärzahlen können leicht in **oktale** und **hexadezimale** Zahlen konvertiert werden
- **2-Komplement** wird benutzt um vorzeichenbehaftete Zahlen darzustellen
 - $b_{n-1}b_{n-2} \dots b_1b_0 = -b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$
- Mit **binären Zahlen** kann man **rechnen** wie in der Schulmathematik.
- Durch limitierte Bitbreiten kann es zu **Überläufen** kommen.
- Rationale Zahlen können als **Gleitkommazahlen** dargestellt werden.
- Was kommt als Nächstes?
 - Wir werden einen **Prozessor** implementieren.
 - Dazu schauen wir uns zunächst die **Grundlagen der Digitaltechnik** an.