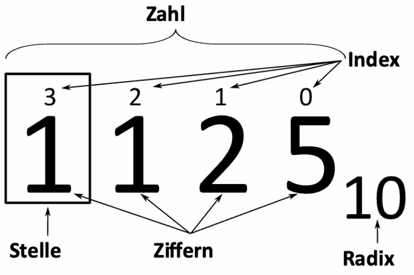
Technische Universität Braunschweig

Informatik für Ingenieure – VL 2

Prof. Dr. Andres Gomez

Zusammenfassung der letzten Sitzung

B-adische Systeme – die Position einer Ziffer bestimmt ihren Wert



Dezimalzahlen: $1011_{10} = 1*10^3 + 0*10^2 + 1*10^1 + 1*10^0$

1000's column
100's column
10's column
1's column

Binärzahlen: $1011_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

8's column
4's column
2's column
1's column

Zusammenfassung der letzten Sitzung

	Wie viele verschiedene Werte?	Wertebereich?
N-stellige Dezimalzahl:	10^N	$[0,10^N-1]$ *
N-stellige Binärzahl:	2^N	$[0,2^N-1]$ *

$B \Rightarrow B^N$

*Bei der Interpretation als Ganzzahl ohne Vorzeichen

Zusammenfassung der letzten Sitzung

	Wie viele verschiedene Werte?	Wertebereich?
N-stellige Dezimalzahl:	10^N	$[0,10^N-1]$ *
N-stellige Binärzahl:	2^N	$[0,2^N-1]$ *

*Bei der Interpretation als Ganzzahl ohne Vorzeichen

$0b0100 = (0100)_2$

Bin	0b0000	0b0001	0b0010	0b0011	0b0100	0b0101	0b0110	0b0111	0b1000	0b1001	0b1010	0b1011	0b1100	0b1101	0b1110	0b1111
Dez.	$(0)_{10}$	$(1)_{10}$	$(2)_{10}$	$(3)_{10}$	$(4)_{10}$	$(5)_{10}$	$(6)_{10}$	$(7)_{10}$	$(8)_{10}$	$(9)_{10}$	$(10)_{10}$	$(11)_{10}$	$(12)_{10}$	$(13)_{10}$	$(14)_{10}$	$(15)_{10}$
Hex.	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF

Hexadezimalzahlen sind eine kürzere Darstellung für lange Binärzahlen

Umwandeln zwischen B-adische Systeme

Zwischen 2er-Potenzen Basen:

Ersetzen Sie zwischen binär, oktal und hexadezimal einfach die entsprechenden Symbole

Umwandeln zwischen B-adische Systeme

Zwischen 2er-Potenzen Basen:

Ersetzen Sie zwischen binär, oktal und hexadezimal einfach die entsprechenden Symbole

Zwischen einer 2er-Potenz und einer Dezimalzahl:

Art 1: Jeweils nach größter noch passender Zweierpotenz suchen

$(7)_{10} = \overset{B_2}{2^2}\ \overset{B_1}{2^1}\ \overset{B_0}{2^0}$

$7 - 111$

$(4)(2)(3)$

Art 2: Durch immer größer werdende Zweierpotenzen dividieren

$7 \div 2 = 3 \quad R\ 1 \quad 2^2$

$3 \div 2 = 1 \quad R\ 1 \quad 2^1$

$1 \div 2 = 1 \quad 2^0$

0b111

Die heutigen Ziele

Kapitel 1:

- Zahlendarstellungen
  - Positive und negative ganze Zahlen
  - Reelle Zahlen
    - Festkommastellung (Fixed-point)
    - Gleitkommastellung (Floating-point)

Die heutigen Ziele

Kapitel 1:

- Zahlendarstellungen
  - Positive und negative ganze Zahlen
  - Reelle Zahlen
    - Festkommastellung (Fixed-point)
    - Gleitkommastellung (Floating-point)

Kapitel 2:

- Einführung Kombinatorische Schaltungen & Boolesche Algebra

Bits, Bytes, Nibbles und Wörter ...

- Bits (Einheit *b*)
  - Höchstwertiges Bit (*msb*)
  - Niedrigstwertiges Bit (*lsb*)

0b 10100010

most significant bit

least significant bit

- Bytes (Einheit *B*) & Nibbles

byte ≙ 8bit

0b 10010110

nibble

Bits, Bytes, Nibbles und Wörter ...

Bits (Einheit *b*)

- Höchstwertiges Bit (*msb*)
- Niedrigstwertiges Bit (*lsb*)

$\underbrace{1}_{\text{most significant bit}}010001\underbrace{0}_{\text{least significant bit}}$

Bytes (Einheit *B*) & Nibbles

$\overbrace{1001\underbrace{0110}_{\text{nibble}}}^{\text{byte}}$

Bytes

- Höchstwertiges Byte (*MSB*)
- Niedrigstwertiges Byte (*LSB*)

word = 32 Bits = 4 Bytes

0x $\overbrace{\underbrace{\text{CE}}_{\text{most significant byte}}\ \text{BF}\ \text{9A}\ \underbrace{\text{D7}}_{\text{least significant byte}}}^{\text{word}}$ 4 bit

8 bits

Doppelwort: 64 Bits = 2 Wörter

Codierung zur Zeichendarstellung

Um Text zu speichern braucht Mann eine Darstellungsform für Zeichen allgemeiner Art

Der ANSI-Code (American National Standards Institute) war ursprünglich ein 7-Bit-Code Bit Code, der die Codierung von 128 Zeichen erlaubt, davon 95 druckbare und 33 Steuerzeichen. erweiterte 8-Bit-ASCII-Zeichensätze wurden später entwickelt

Codierung zur Zeichendarstellung

Um Text zu speichern braucht Mann eine Darstellungsform für Zeichen allgemeiner Art

Der ANSI-Code (American National Standards Institute) war ursprünglich ein 7-Bit-Code Bit Code, der die Codierung von 128 Zeichen erlaubt, davon 95 druckbare und 33 Steuerzeichen. erweiterte 8-Bit-ASCII-Zeichensätze wurden später entwickelt

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

A = 0x41

a = 61

Moderne Systeme verwenden eine größere UTF-8 Darstellung, die viel mehr Zeichen für verschiedene Sprachfamilien enthält und braucht 1-4 Bytes pro Zeichen.

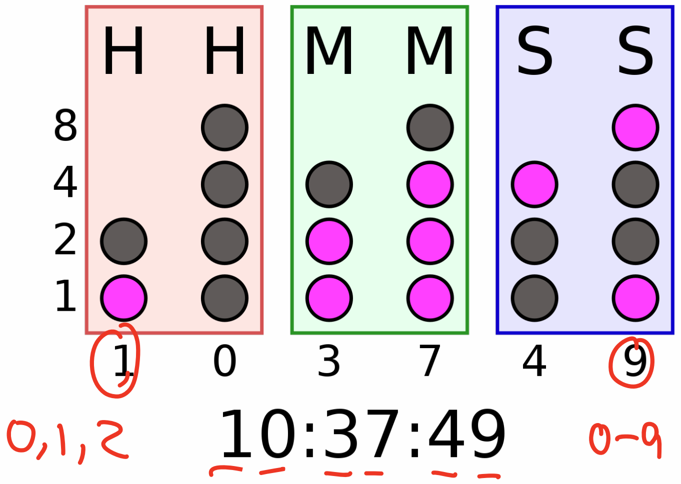
Binary Coded Decimal

Bei BCD wird jede Dezimalziffer durch einen 4-bit Binärcode dargestellt.

das Gewicht jeder Bitposition entspricht einer Potenz von 10.

Häufig verwendet in elektronischen Geräten, die Dezimalzahlen anzeigen müssen, z. B. in Digitaluhren und Taschenrechnern.

Im Vergleich zu binären Positionssystemen zeichnet sich BCD durch eine genauere Darstellung und Rundung von Dezimalmengen aus.



Zweierpotenzen und Präfixe

10^3 = 1 Kilo	(K)	= 1,000
10^6 = 1 Mega	(M)	= 1,000,000
10^9 = 1 Giga	(G)	= 1,000,000,000
2^{10} = 1 Kilo	(Ki)	≈ 1000 (1,024)
2^{20} = 1 Mega	(Mi)	≈ 1 Million (1,048,576)
2^{30} = 1 Giga	(Gi)	≈ 1 Milliarde (1,073,741,824)

Beispiel:

4 GiB: Maximal adressierbare Speichergröße für 32b-Prozessoren

Rechnen wir ein bisschen nach

Was ist der Wert von 2^{24}? $2^4 \cdot 2^{20} = 16 \cdot Mi$

Wie viele verschiedene Werte kann eine 32b Variableannehmen? $2^2 \cdot 2^{30} = 4\ Gi$

Rechnen wir ein bisschen nach

Was ist der Wert von 2^{24}?

Wie viele verschiedene Werte kann eine 32b Variableannehmen?

Addieren wir zwei 4-stellige Zahlen:

Dezimal:

```
   1 1
   3734
+  5168
-------
   8902
```

Binär:

```
      1 1 1
   0b1001
+  0b0111
---------
Ü:1  0000
```

Überlauf

Digitale Systeme arbeiten mit einer **festen** Anzahl an Bits

Eine Addition **läuft über**, wenn ihr Ergebnis nicht mehr in die verfügbare Anzahl von Bits hineinpasst

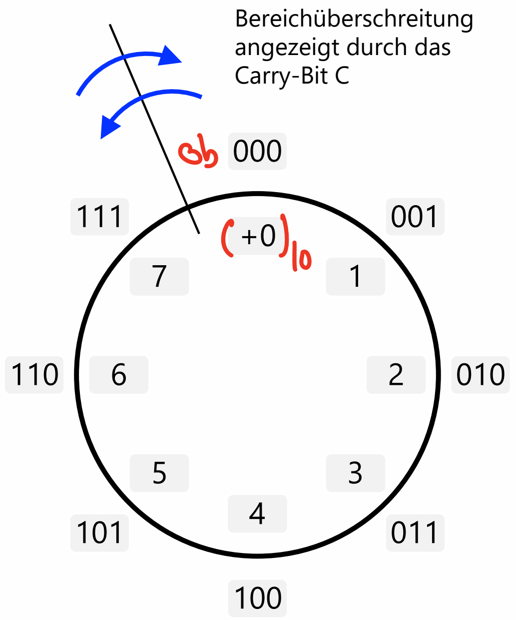
Beispiel: 0b111 + 0b010, gerechnet mit 3 Bit Breite

Überlauf

Digitale Systeme arbeiten mit einer **festen** Anzahl an Bits

Eine Addition **läuft über**, wenn ihr Ergebnis nicht mehr in die verfügbare Anzahl von Bits hineinpasst

Beispiel: 0b111 + 0b010, gerechnet mit 3 Bit Breite



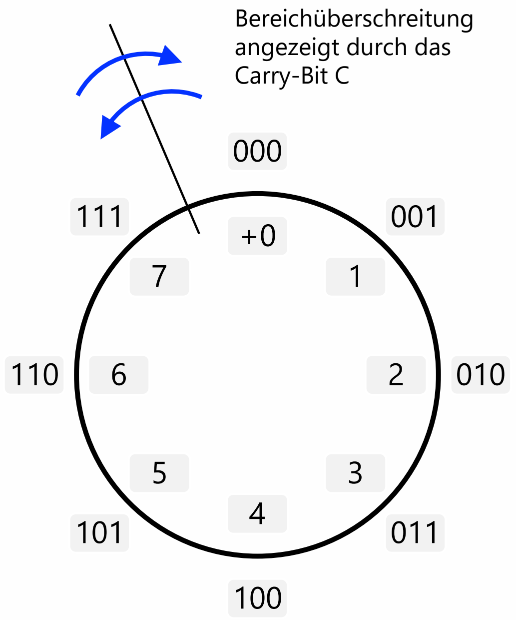
Bereichsüber- oder -unterschreitungen können in eine zusätzliche Stelle (Carry-Bit) erkannt werden

Überlauf

Digitale Systeme arbeiten mit einer **festen** Anzahl an Bits

Eine Addition **läuft über**, wenn ihr Ergebnis nicht mehr in die verfügbare Anzahl von Bits hineinpasst

Beispiel: 0b111 + 0b010, gerechnet mit 3 Bit Breite



Bereichsüber- oder -unterschreitungen können in eine zusätzliche Stelle (Carry-Bit) erkannt werden

Wir haben nur von positiven Anzahlen geredet. Wozu mit den **negativen Zahlen**?

Vorzeichenbehaftete Binärzahlen

Erste Idee: 1 Bit fürs **Vorzeichnen**, n-1 Bits für den **Betrag**

Vorzeichenbit ist höchstwertiges Bit (msb)

Positive Zahl: Vorzeichenbit = 0
Negative Zahl: Vorzeichenbit = 1

$$N: \{\boldsymbol{b_{n-1}}, \boldsymbol{b_{n-2}}, \dots, \boldsymbol{b_1}, \boldsymbol{b_0}\}$$

$$N = (-1)^{\boldsymbol{b_{n-1}}} \sum_{i=0}^{n-2} \boldsymbol{b_i} \cdot 2^i$$

Vorzeichenbehaftete Binärzahlen

Erste Idee: 1 Bit fürs **Vorzeichnen**, n-1 Bits für den **Betrag**

Vorzeichenbit ist höchstwertiges Bit (msb)

Positive Zahl: Vorzeichenbit = 0
Negative Zahl: Vorzeichenbit = 1

$$N: \{\boldsymbol{b_{n-1}}, \boldsymbol{b_{n-2}}, \dots, \boldsymbol{b_1}, \boldsymbol{b_0}\}$$

$$N = (-1)^{\boldsymbol{b_{n-1}}} \sum_{i=0}^{n-2} \boldsymbol{b_i} \cdot 2^i$$

Beispiel:

4-bit Vorzeichen/Betrag-Darstellung von ± 6:

+6 = 0b**0110**

- 6 = 0b**1110**

Vorzeichenbehaftete Binärzahlen

Erste Idee: 1 Bit fürs **Vorzeichnen**, n-1 Bits für den **Betrag**

Vorzeichenbit ist höchstwertiges Bit (msb)

Positive Zahl: Vorzeichenbit = 0
Negative Zahl: Vorzeichenbit = 1

$$N:\{\boldsymbol{b_{n-1}}, \boldsymbol{b_{n-2}}, \dots, \boldsymbol{b_1}, \boldsymbol{b_0}\}$$

$$N = (-1)^{\boldsymbol{b_{n-1}}} \sum_{i=0}^{n-2} \boldsymbol{b_i} \cdot 2^i$$

Beispiel:

4-bit Vorzeichen/Betrag-Darstellung von ± 6:

+6 = 0b**0110**

- 6 = 0b**1110**

Wertebereich einer Zahl in Vorzeichen/Betrag-Darstellung : $[-(2^{n-1}-1), 2^{n-1}-1]$

Darstellung als Vorzeichen/Betrag: Probleme

Addition schlägt fehl
Beispiel: -3 + 3:

```
      0b111
    + 0b011
Ü=1  0b010 (falsch!)
```

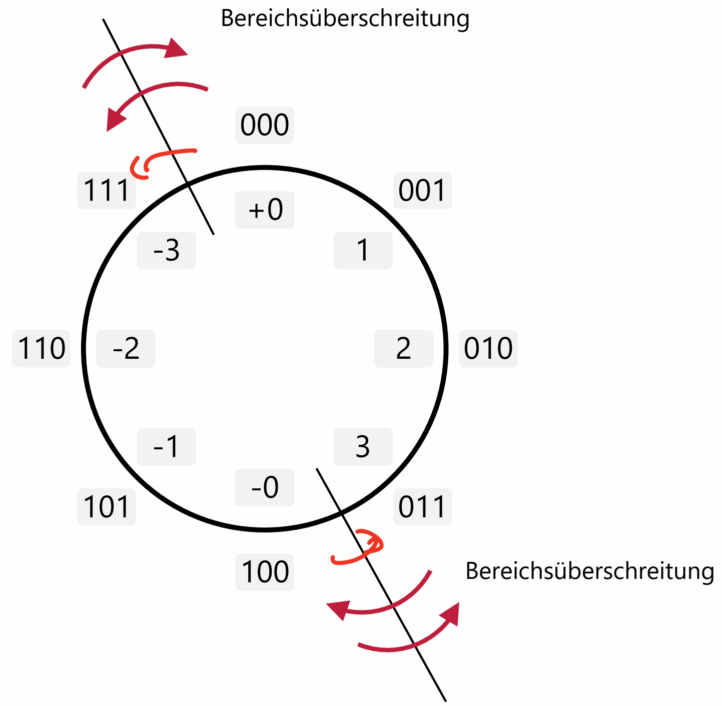
Zwei Darstellungen für Null:

0b1000 = -(0)
0b0000 = +(0)

Darstellung als Vorzeichen/Betrag: Probleme

Addition schlägt fehl
Beispiel: -3 + 3:

0b111
+ 0b011
Ü=1 0b010 (falsch!)



Zwei Darstellungen für Null:

0b1000 = -(0)
0b0000 = +(0)

In diesem Code wachsen die Zahlen nicht gleichmäßig

Andere Negative Darstellungen

Offsetdarstellung (Exzesscode) verschiebt den Zahlenbereich durch Addition

Codierung	Verschiebung	Code							
		000	001	010	011	100	101	110	111
Exzess-0	0	0	1	2	3	4	5	6	7
Exzess-1	1	−1	0	1	2	3	4	5	6
Exzess-2	2	−2	−1	0	1	2	3	4	5
Exzess-3	3	−3	−2	−1	0	1	2	3	4
Exzess-4	4	−4	−3	−2	−1	0	1	2	3

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

$$\begin{array}{rl} 9999 & 10^1 - 1 \\ -0005 & \text{Zahl} \\ \hline 9994 & \text{EK} \end{array}$$

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

9999	$10^1 - 1$
-0005	Zahl
9994	EK

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

0b1111	$2^3 - 1$
-0b0101	Zahl
0b1010	EK

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

9999	$10^1 - 1$
-0005	Zahl
9994	EK

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

0b1111	$2^3 - 1$
-0b0101	Zahl
0b1010	EK

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Ist EK eindeutig?

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

9999	$10^1 - 1$
-0005	Zahl
9994	EK

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

0b1111	$2^3 - 1$
-0b0101	Zahl
0b1010	EK

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Ist EK eindeutig? EK(+0) ≠ EK(-0)

```
+0 in EK: 0b0000
-0 in EK: 0b1111
```

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

9999	$10^1 - 1$
-0005	Zahl
9994	EK

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

0b1111	$2^3 - 1$
-0b0101	Zahl
0b1010	EK

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Ist EK eindeutig? EK(+0) $\neq$ EK(-0)

```
+0 in EK: 0b0000
-0 in EK: 0b1111
```

Wenn wir 1 addieren, können wir diesen Grenzfall unterscheiden

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

$$\begin{array}{rl} 9999 & 10^1 - 1 \\ -0005 & \text{Zahl} \\ \hline 9994 & \text{EK} \end{array}$$

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

$$\begin{array}{rl} \texttt{0b1111} & 2^3 - 1 \\ -\texttt{0b0101} & \text{Zahl} \\ \hline \texttt{0b1010} & \text{EK} \end{array}$$

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Ist EK eindeutig? EK(+0) ≠ EK(-0)

```
+0 in EK: 0b0000
-0 in EK: 0b1111
```

Wenn wir 1 addieren, können wir diesen Grenzfall unterscheiden

Zweierkomplement (ZK)
entspricht Ergänzung auf B^n

Beispiel: wir wollen -5 in ZK darstellen (Vier Ziffern):

$$\begin{array}{rl} \texttt{0b1010} & \text{EK} \\ +\texttt{0b0001} & 1 \\ \hline \texttt{0b1011} & \text{ZK} \end{array}$$

```
+5 in ZK: 0b0101
-5 in ZK: 0b1011
```

Komplementdarstellung

Einerkomplement (EK)
entspricht Ergänzung auf B^n-1

9999	$10^1 - 1$
−0005	Zahl
9994	EK

Beispiel: wir wollen -5 in EK darstellen (Vier Ziffern):

0b1111	$2^3 - 1$
−0b0101	Zahl
0b1010	EK

```
+5 in EK: 0b0101
-5 in EK: 0b1010
```

Ist EK eindeutig? EK(+0) ≠ EK(-0)

```
        1 1 1
+0 in EK: 0b0000
-0 in EK: 0b1111
          + 0001
Ü1          0000
```

Wenn wir 1 addieren, können wir diesen Grenzfall unterscheiden

Zweierkomplement (ZK)
entspricht Ergänzung auf B^n

Eindeutig: ZK(+0) = ZK(-0)
= 0b0000

Beispiel: wir wollen -5 in ZK darstellen (Vier Ziffern):

0b1010	EK
+0b0001	1
0b1011	ZK

```
+5 in ZK: 0b0101
-5 in ZK: 0b1011
```

Zahlendarstellung im Zweierkomplement

Wie vorzeichenlose Binärdarstellung, aber … msb hat nun einen Wert von -2^{n-1}

$$N = b_{n-1} \cdot \ (2^{n-1}) \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Größte positive 4b Zahl: 0b0111 = $2^2 + 2^1 + 2^0 = 7$

Kleinste negative 4b Zahl: 0b1000 = $-2^3 = -8$

Zahlendarstellung im Zweierkomplement

Wie vorzeichenlose Binärdarstellung, aber … msb hat nun einen Wert von -2^{n-1}

$$N = b_{n-1} \cdot (2^{n-1}) \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Größte positive 4b Zahl: 0b0111 = $2^2 + 2^1 + 2^0 = 7$

Kleinste negative 4b Zahl: 0b1000 = $-2^3 = -8$

0b1111 = -1

Wertebereich einer *N*-bit Zweierkomplementzahl: **[-(2^{N-1}), 2^{N-1}-1]**

Zahlendarstellung im Zweierkomplement

Wie vorzeichenlose Binärdarstellung, aber … msb hat nun einen Wert von -2^{n-1}

$$N = b_{n-1} \cdot (2^{n-1}) \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Größte positive 4b Zahl: 0b0111 = $2^2 + 2^1 + 2^0 = 7$

Kleinste negative 4b Zahl: 0b1000 = $-2^3 = -8$

Wertebereich einer *N*-bit Zweierkomplementzahl: **[$-(2^{N-1})$, $2^{N-1}-1$]**

msb gibt immer noch das Vorzeichen an

negativ → $b_{n-1} = 1$
positive → $b_{n-1} = 0$

Zahlendarstellung im Zweierkomplement

Wie vorzeichenlose Binärdarstellung, aber … msb hat nun einen Wert von -2^{n-1}

$$N = b_{n-1} \cdot \left(2^{n-1}\right) \sum_{i=0}^{n-2} b_i \cdot 2^i$$

Größte positive 4b Zahl: 0b0111 = $2^2 + 2^1 + 2^0 = 7$

Kleinste negative 4b Zahl: 0b1000 = $-2^3 = -8$

Wertebereich einer *N*-bit Zweierkomplementzahl: **$[-(2^{N-1}), 2^{N-1}-1]$**

msb gibt immer noch das Vorzeichen an

negativ → b_{n-1} = 1
positive → b_{n-1} = 0

Addition liefert wieder korrekte Ergebnisse! → Subtraktion ist ***die Summe*** eines Negatives Zahl

Wir können die gleiche Hardware-Einheit zum Addieren und Subtrahieren verwenden

Zweierkomplement arithmetisch bilden

Algorithmus:

Alle Bits invertieren (0→1, 1→0)

Dann 1 addieren

In ***beide*** Richtungen anwendbar

Vorzeichenwechsel: $k \rightarrow -k$

5bit

Beispiel: Vorzeichenwechsel von $(3)_{10}$ = 0b00011

0b11100 Ek

\+ 1

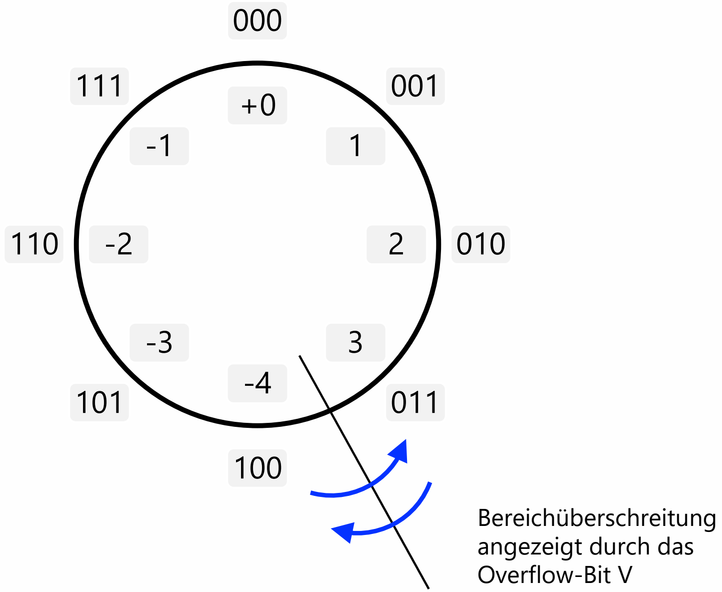
0b11101

Beispiel: Vorzeichenwechsel von $(-3)_{10}$ = 0b11101 ⇒

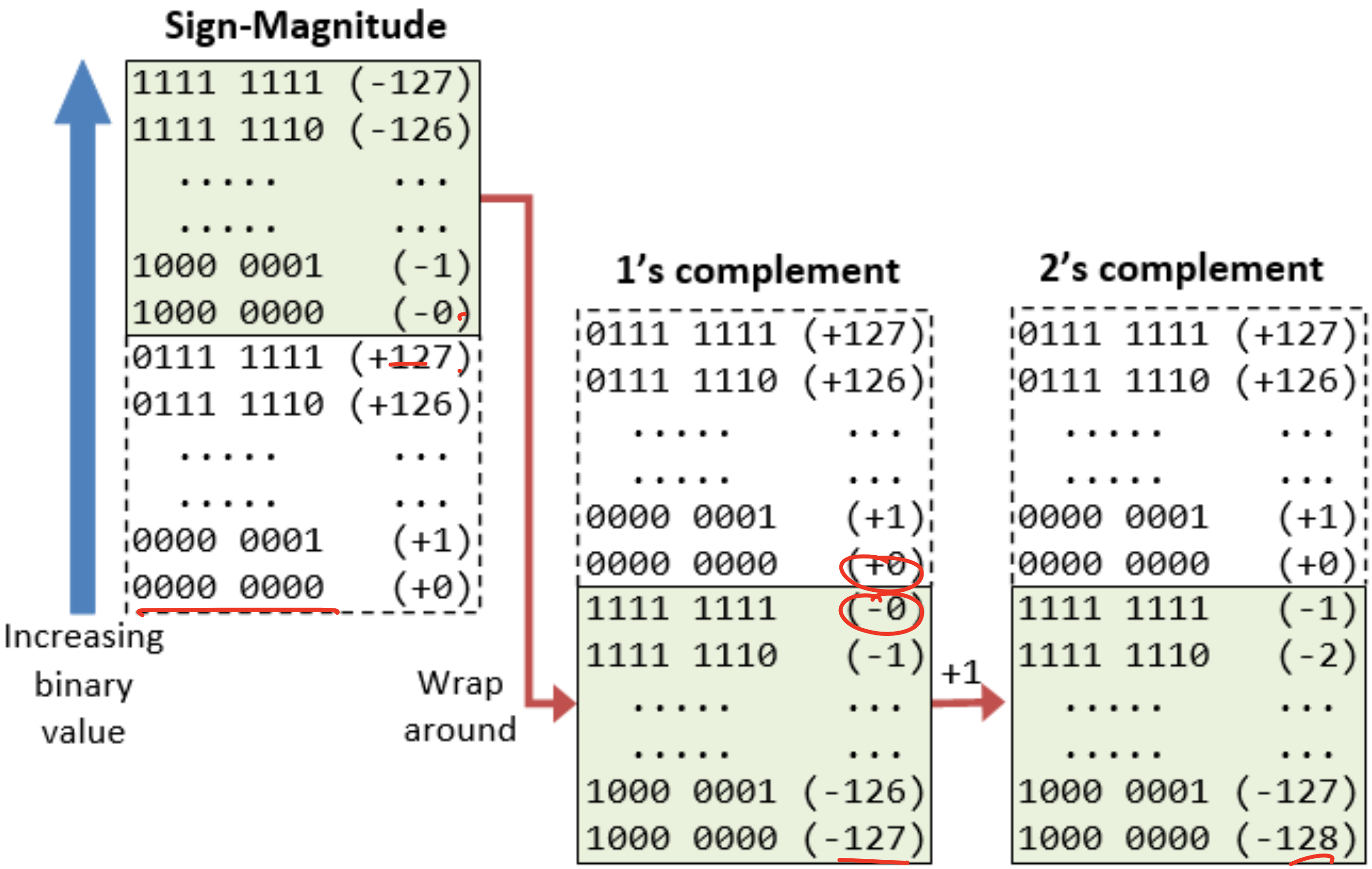
0b00010 Ek

\+ 1

0b00011



Visualisierung verschiedener Darstellungen



Zweierkomplement

Nachteile: Konvertierung erfordert ein „carry"
Überläufe werden nicht automatisch erkannt.

Fallunterscheidung bei Addition im 2er Komplement:

Fall 1:

```
        0b0000 0001   ≡ (+1)10
      + 0b1111 1111   ≡ (-1)10
Ü=1     0b0000 0000   ≡ ( 0)10     √
```

Zweierkomplement

Nachteile: Konvertierung erfordert ein „carry"
Überläufe werden nicht automatisch erkannt.

Fallunterscheidung bei Addition im 2er Komplement:

Fall 1:

```
        0b0000 0001   ≡  (+1)10
      + 0b1111 1111   ≡  (-1)10
Ü=1     0b0000 0000   ≡  ( 0)10      √
```

Fall 2:

```
        0b1111 1111   ≡  (-1)10
      + 0b1111 1110   ≡  (-2)10
Ü=1     0b1111 1101   ≡  (-3)10      √
```

Zweierkomplement

Nachteile: Konvertierung erfordert ein „carry"
Überläufe werden nicht automatisch erkannt.

Fallunterscheidung bei Addition im 2er Komplement:

Fall 1:

	0b0000 0001	≡	$(+1)_{10}$	
	+ 0b1111 1111	≡	$(-1)_{10}$	
Ü=1	0b0000 0000	≡	$(\ 0)_{10}$	√

Fall 2:

	0b1111 1111	≡	$(-1)_{10}$	
	+ 0b1111 1110	≡	$(-2)_{10}$	
Ü=1	0b1111 1101	≡	$(-3)_{10}$	√

Fall 3:

	0b0000 0001	≡	$(+1)_{10}$	
	+ 0b0000 0010	≡	$(+2)_{10}$	
Ü=0	0b0000 0011	≡	$(+3)_{10}$	√

Zweierkomplement

Fallunterscheidung bei Addition im 2er Komplement:

Fall 4:

$$
\begin{array}{rrcl}
 & \texttt{0b0000 0001} & \equiv & (+1)_{10} \\
 & + \ \texttt{0b0111 1111} & \equiv & (+127)_{10} \\
\hline
\ddot{U}=0 & \texttt{0b1000 0000} & \equiv & (-128)_{10}
\end{array}
$$

Falsch

Zweierkomplement

Fallunterscheidung bei Addition im 2er Komplement:

Fall 4:

$$
\begin{array}{lrcl}
 & \texttt{0b0000 0001} & \equiv & (+1)_{10} \\
 & + \ \texttt{0b0111 1111} & \equiv & (+127)_{10} \\
\hline
Ü=0 & \texttt{0b1000 0000} & \equiv & (-128)_{10}
\end{array}
$$

Falsch

Fall 5:

$$
\begin{array}{lrcl}
 & \texttt{0b1000 0000} & \equiv & (-128)_{10} \\
 & + \ \texttt{0b1111 1111} & \equiv & (-1)_{10} \\
\hline
Ü=1 & \texttt{0b0111 1111} & \equiv & (+127)_{10}
\end{array}
$$

Falsch

Zweierkomplement

Fallunterscheidung bei Addition im 2er Komplement:

Fall 4:

	0b0000 0001	≡	$(+1)_{10}$
	+ 0b0111 1111	≡	$(+127)_{10}$
Ü=0	0b1000 0000	≡	$(-128)_{10}$ **Falsch**

Fall 5:

	0b1000 0000	≡	$(-128)_{10}$
	+ 0b1111 1111	≡	$(-1)_{10}$
Ü=1	0b0111 1111	≡	$(+127)_{10}$ **Falsch**

Erkennung der Bereichsüberschreitung:

Wenn die obersten Bitstellen beider Summanden gleich sind und sich von der obersten Bitstelle in der Summe unterscheiden, ist eine Bereichsüberschreitung aufgetreten (Over- / Underflow)

Expandieren und Trunkieren

Damit mathematische Operationen korrekt ausgeführt werden können, müssen zwei Operanden die gleichen Bits und Formate haben. Andernfalls müssen wir expandieren, trunkieren oder „Typecasten".

Expandieren und Trunkieren

Damit mathematische Operationen korrekt ausgeführt werden können, müssen zwei Operanden die gleichen Bits und Formate haben. Andernfalls müssen wir expandieren, trunkieren oder „Typecasten“.

Zu expandieren müssen wir die Anzahl Bits der schmaleren Zahl auf die andere Zahl erhöhen:

Zwei Möglichkeiten: Auffüllen mit dem bisherigen Vorzeichen (sign extension)
Auffüllen mit führenden Nullen (zero extension)

Expandieren und Trunkieren

Damit mathematische Operationen korrekt ausgeführt werden können, müssen zwei Operanden die gleichen Bits und Formate haben. Andernfalls müssen wir expandieren, trunkieren oder „Typecasten“.

Zu expandieren müssen wir die Anzahl Bits der schmaleren Zahl auf die andere Zahl erhöhen:

Zwei Möglichkeiten: Auffüllen mit dem bisherigen Vorzeichen (sign extension)
Auffüllen mit führenden Nullen (zero extension)

Beim Trunkieren schneiden wir Bits ab und interpretieren die Zahl neu.
Es ist möglich, Informationen zu verlieren.

Auffüllen mit dem bisherigen Vorzeichen

Vorzeichenbit nach links kopieren bis gewünschte Breite erreicht

Zahlenwert bleibt unverändert

Auch bei negativen Zahlen!

Beispiel 1:

4-bit Darstellung von 3 = 0b0011

8-bit aufgefüllt durch Vorzeichen: 0b0000 0011

Auffüllen mit dem bisherigen Vorzeichen

Vorzeichenbit nach links kopieren bis gewünschte Breite erreicht

Zahlenwert bleibt unverändert

Auch bei negativen Zahlen!

Beispiel 1:

4-bit Darstellung von 3 = 0b0011

8-bit aufgefüllt durch Vorzeichen:

Beispiel 2:

4-bit Darstellung von -5 = 0b1011

8-bit aufgefüllt durch Vorzeichen : 0b 1111 1011

Erweitern durch Auffüllen mit Nullbits

Nullen nach links anhängen bis gewünschte Breite erreicht

Zerstört Wert von negativen Zahlen

Positive Zahlen bleiben unverändert

Beispiel 1:

4-bit Wert = 0b0011 = $(3)_{10}$

8-bit durch Auffüllen mit Nullbits: 0b0000 0011

Erweitern durch Auffüllen mit Nullbits

Nullen nach links anhängen bis gewünschte Breite erreicht

Zerstört Wert von negativen Zahlen

Positive Zahlen bleiben unverändert

Beispiel 1:

4-bit Wert = 0b0011 = $(3)_{10}$

8-bit durch Auffüllen mit Nullbits:

Beispiel 2:

4-bit Wert = 0b1011 = $(-5)_{10}$

8-bit durch Auffüllen mit Nullbits: 0b0000 1011

Noch ein letztes Wort zum Thema Umwandlung

Konvertierung zwischen vorzeichenbehafteten ←→ Vorzeichenlose Zahlen ist auch nicht trivial

Die Vorzeichenumwandlung ändert den Kontext,
behält aber normalerweise das gleiche Bitmuster bei.

Noch ein letztes Wort zum Thema Umwandlung

Konvertierung zwischen vorzeichenbehafteten ←→ Vorzeichenlose Zahlen ist auch nicht trivial

Die Vorzeichenumwandlung ändert den Kontext,
behält aber normalerweise das gleiche Bitmuster bei.

Was passiert, wenn wir die größtmögliche
vorzeichenlose 8-Bit-Zahl in eine vorzeichenbehaftete
8-Bit-Ganzzahl umwandeln?

Noch ein letztes Wort zum Thema Umwandlung

Konvertierung zwischen vorzeichenbehafteten ←→ Vorzeichenlose Zahlen ist auch nicht trivial

Die Vorzeichenumwandlung ändert den Kontext, behält aber normalerweise das gleiche Bitmuster bei.

Was passiert, wenn wir die größtmögliche vorzeichenlose 8-Bit-Zahl in eine vorzeichenbehaftete 8-Bit-Ganzzahl umwandeln?

Fehler in C++ StdLib (2012)

Binary	Unsigned	Signed
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Darstellung von ganzen Zahlen

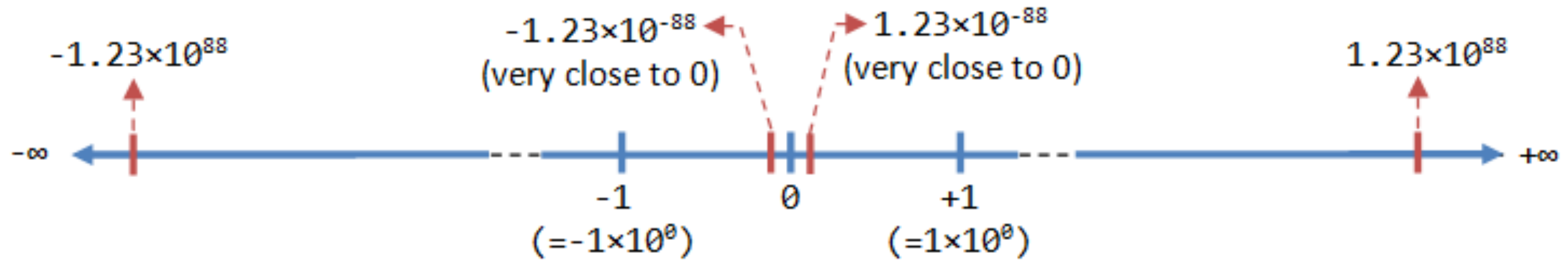
Wortbreite	Vorzeichenlos	Vorzeichen/Betrag	Zweierkomplement
8 bit	0 ... 255	-127 ... -0 , +0 ... +127	-128 ... 0 ... +127
16 bit	0 ... 65535	-32767 ... -0 , +0 ... +32767	-32768 ... 0 ... +32267
32 bit	0 ... 4Gi - 1	-(2Gi-1) ... -0 , +0 ... +(2Gi-1)	-2Gi ... 0 ... +2Gi-1
n bit	0 ... 2^{n} - 1	$-(2^{n-1}-1)$ ... -0 , +0 ... $+(2^{n-1}-1)$	-2^{n-1} ... 0 ... $+2^{n-1-}1$

Darstellung von reellen Zahlen

Reelle Zahlen sind kontinuierlich - sie können unendlich viele Werte haben.

Dezimalzahlen mit Gleitkomma können sehr große positive Zahlen, negative Zahlen und Zahlen nahe Null darstellen.

$1230000 \rightarrow 0$

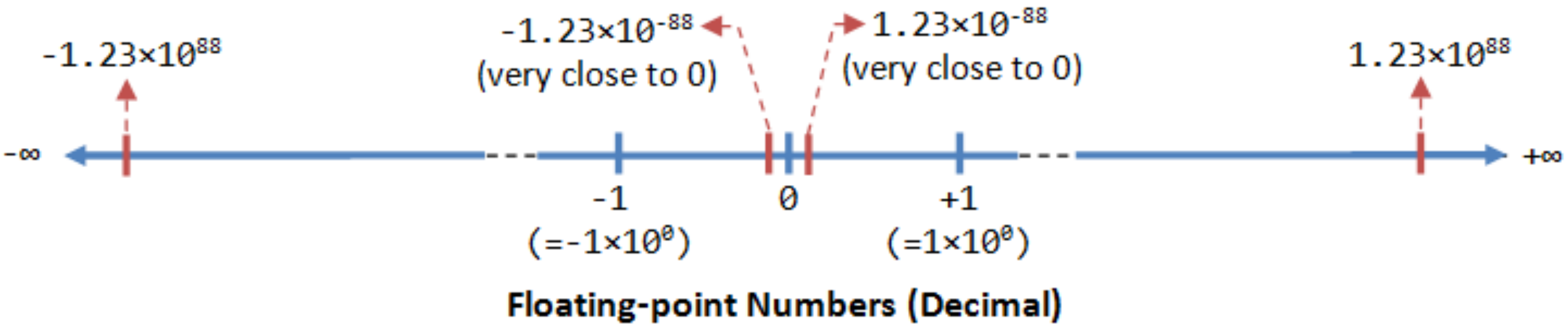


Floating-point Numbers (Decimal)

Darstellung von reellen Zahlen

Reelle Zahlen sind kontinuierlich - sie können unendlich viele Werte haben.

Dezimalzahlen mit Gleitkomma können sehr große positive Zahlen, negative Zahlen und Zahlen nahe Null darstellen.



Wie kann man reelle Zahlen in Binärform darstellen?

Festkommaformat (Fixed-Point)

Vorgegebene Anzahl von Bits vor
und nach dem Komma:

$$\mathbf{b_{m-1}}, \mathbf{b_{m-2}}, \dots, \mathbf{b_1}, \mathbf{b_0} \bullet \boldsymbol{b_{-1}}, \boldsymbol{b_{-2}}, \dots, \boldsymbol{b_{-n}}$$

$$N = \sum_{i=-n}^{m} b_i \cdot 2^i$$

Festkommaformat (Fixed-Point)

Vorgegebene Anzahl von Bits vor
und nach dem Komma:

$$\mathbf{b_{m-1}, b_{m-2}, \dots, b_1, b_0} \bullet \boldsymbol{b_{-1}, b_{-2}, \dots, b_{-n}}$$

$$N = \sum_{i=-n}^{m} b_i \cdot 2^i$$

Beispiel:

$0b\mathbf{1100}.\mathbf{0011} = \mathbf{2^3} + \mathbf{2^2} + \mathbf{2^{-3}} + \mathbf{2^{-4}} = (12.1875)_{10}$

2^{-1} 2^{-2} 2^{-3} 2^{-4}

Einer **4**-**4** Fixed-Point Zahl hat:

4 bit für den **Ganzzahligenteil**

4 bit für den **Nachkommateil**

Festkommaformat (Fixed-Point)

Vorgegebene Anzahl von Bits vor
und nach dem Komma:

$$\mathbf{b_{m-1}, b_{m-2}, \dots, b_1, b_0} \bullet \boldsymbol{b_{-1}, b_{-2}, \dots, b_{-n}}$$

$$N = \sum_{i=-n}^{m} b_i \cdot 2^i$$

Beispiel:

0b**1100**.**0011** = $\mathbf{2^3} + \mathbf{2^2} + \mathbf{2^{-3}} + \mathbf{2^{-4}}$ = 12.1875

Einer **4**-**4** Fixed-Point Zahl hat:

4 bit für den **Ganzzahligenteil**

4 bit für den **Nachkommateil**

Durch diese Darstellung kann man gebrochene Zahlen bis auf die durch die Nachkommastellen des zweiten Wortes gegebene Genauigkeit erfassen.

Arithmetik in Festkommadarstellung lässt sich einfach implementieren, weist aber auch die entsprechenden Beschränkungen in Zahlenbereich und Genauigkeit auf.

Gleitkommadarstellung (floating point)

Nehmen wir an, wir haben die Nummer $5 \cdot 2^{100}$

Für die Darstellung im Festkommaformat wären 103 Binärziffern erforderlich.
Eine kompaktere Darstellung wäre die Form $X \cdot 2^{E}$

Gleitkommadarstellung (floating point)

Nehmen wir an, wir haben die Nummer $5{\cdot}2^{100}$

Für die Darstellung im Festkommaformat wären 103 Binärziffern erforderlich.
Eine kompaktere Darstellung wäre die Form $X \cdot 2^{E}$

Drei Zahlen sind nötig Vorzeichen V, Signifikant S und Exponent E

$$N = (-1)^{V} \cdot S \cdot B^{E}$$

Auch geschrieben als: $\mathrm{V} \cdot S \cdot B^{E}$

Festkomma: 0b$1\underbrace{0110010}_{\text{7 Stellen}}$.001

Gleitkomma: $0b1.0110010001{\cdot}2^{7}$

Gleitkommadarstellung (floating point)

Nehmen wir an, wir haben die Nummer $5 \cdot 2^{100}$

Für die Darstellung im Festkommaformat wären 103 Binärziffern erforderlich.
Eine kompaktere Darstellung wäre die Form $X \cdot 2^E$

Drei Zahlen sind nötig Vorzeichen V, Signifikant S und Exponent E

$$N = (-1)^V \cdot S \cdot B^E$$

Auch geschrieben als: $\mathrm{V} \cdot S \cdot B^E$

Festkomma: 0b10110010.001

7 Stellen

Gleitkomma: $0b1.0110010001 \cdot 2^{7}$

Die Anzahl m der Nachkommastellen bestimmt die Genauigkeit (2^{-m}), und der Wertebereich des Zweierexponenten E legt den Wertebereich (Zahlenraum) der Gleitkommazahl zu $2E$ fest.

Mathematische Operationen mit Gleitkomma

Die Multiplikation von Gleitkommazahlen ist einfach, denn es gilt:

$$X \cdot Y = V_x \cdot S_x \cdot B^{E_x} \cdot V_y \cdot S_y \cdot B^{E_y} = (V_x \cdot V_y) \cdot (S_x \cdot S_y) \cdot B^{E_x + E_y}$$

Danach werden die Signifikanten $s = g,b$ (Bruchteile b einschließlich der Vorkommastellen g) als normale Zahlen multipliziert, und die Exponenten $e = E - E0$ werden addiert.

Entsprechendes gilt für die Division.

Mathematische Operationen mit Gleitkomma

Die Multiplikation von Gleitkommazahlen ist einfach, denn es gilt:

$$X \cdot Y = V_x \cdot S_x \cdot B^{E_x} \cdot V_y \cdot S_y \cdot B^{E_y} = (V_x \cdot V_y) \cdot (S_x \cdot S_y) \cdot B^{E_x + E_y}$$

Danach werden die Signifikanten *s* = *g*,*b* (Bruchteile *b* einschließlich der Vorkommastellen *g*) als normale Zahlen multipliziert, und die Exponenten *e* = *E* – *E*0 werden addiert.

Entsprechendes gilt für die Division.

Schwieriger ist dagegen die Addition: Hier müssen die Exponenten zunächst angeglichen werden, denn eine Addition der Signifikanten kann nur dann stellenrichtig ausgeführt werden, wenn die Exponenten gleich sind:

$7.87 \times 10^{-6} + 5.637 \times 10^{-3}$

$$X + Y = V_x \cdot S_x \cdot B^{E_x} + V_y \cdot S_y \cdot B^{E_y} = \left((V_y \cdot S_x) + (V_y \cdot \boldsymbol{S'_y})\right) \cdot B^{E_x}$$

Videobeispiel

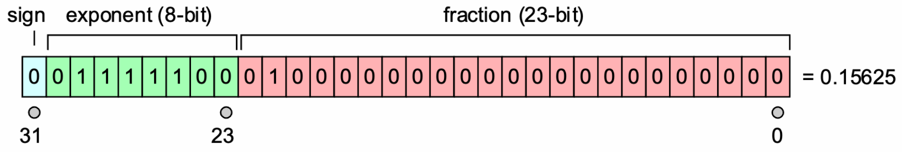
IEEE 754 Gleitkommadarstellungformat

$$R = (-1)^v \cdot s \cdot 2^e = (-1)^v \cdot g.b \cdot 2^e$$

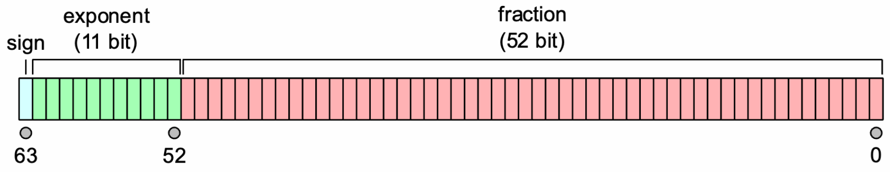
IEEE 754 Gleitkommadarstellungformat

$$R = (-1)^v \cdot s \cdot 2^e = (-1)^v \cdot g.b \cdot 2^e$$

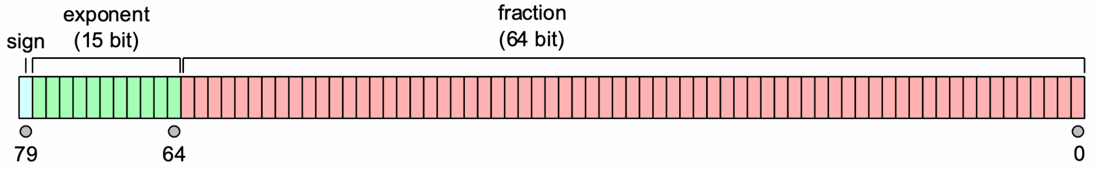
Einfache Genauigkeit



Doppelte Genauigkeit



Erweiterte Genauigkeit



IEEE 754 Gleitkommadarstellungformat

$$R = (-1)^v \cdot s \cdot 2^e = (-1)^v \cdot g.b \cdot 2^e$$

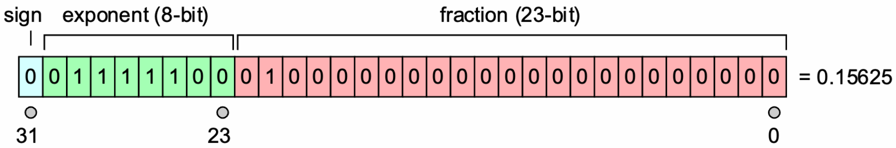
Für $\mathrm{E} \geq 1$: $g = 1$ (normalisiert)

Exponent: $e = E - E_0$

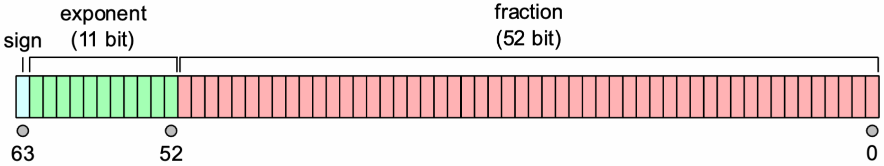
Exzesscode, mit E_0 abhängig von der Präzision

Für $\mathrm{E} = 0$: $g = 0$ (denormalisiert)

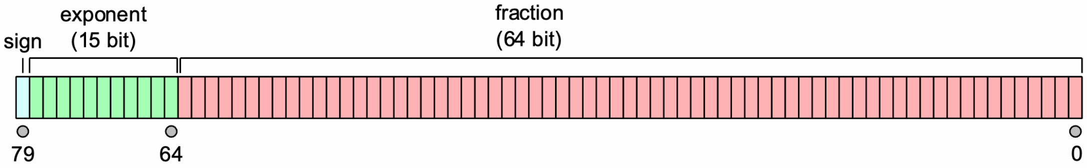
Einfache Genauigkeit



Doppelte Genauigkeit



Erweiterte Genauigkeit



IEEE 754 Erläutert

1. Normalisiert („leading 1“):

   Einfache Genauigkeit:

   $E_0 = 2^{k-1} - 1 = 127$

   $e \in [-126,127]$

X 0b0000 0000

s	≠ 0 and ≠ 255	f
1 bit	8 bits	23 bits

X 0b1111 1111

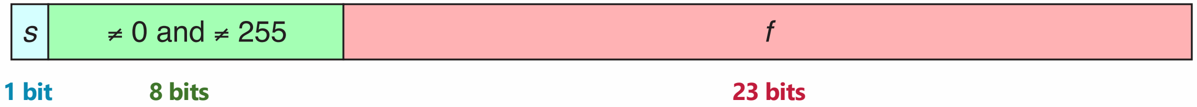
IEEE 754 Erläutert

1. Normalisiert („leading 1"):

    Einfache Genauigkeit:

    $E_0 = 2^{k-1} - 1 = 127$

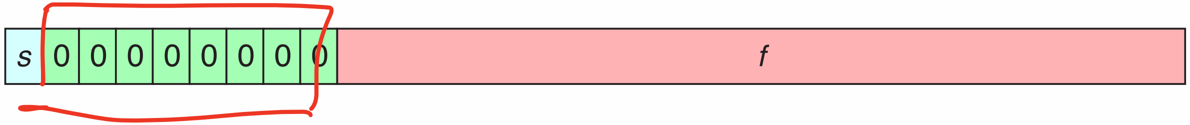
    $e \in [-126{,}127]$



2. Denormalisiert

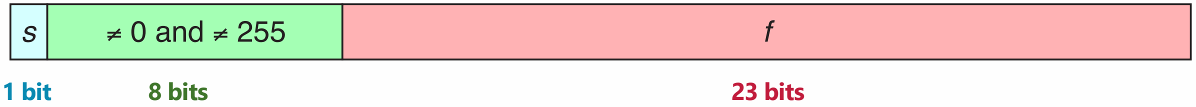
    Wichtig für die Darstellung von 0.

    Zahlen in der Nähe von 0



IEEE 754 Erläutert

1. Normalisiert („leading 1“):
   Einfache Genauigkeit:
   $E_0 = 2^{k-1} - 1 = 127$
   $e \in [-126,127]$



2. Denormalisiert

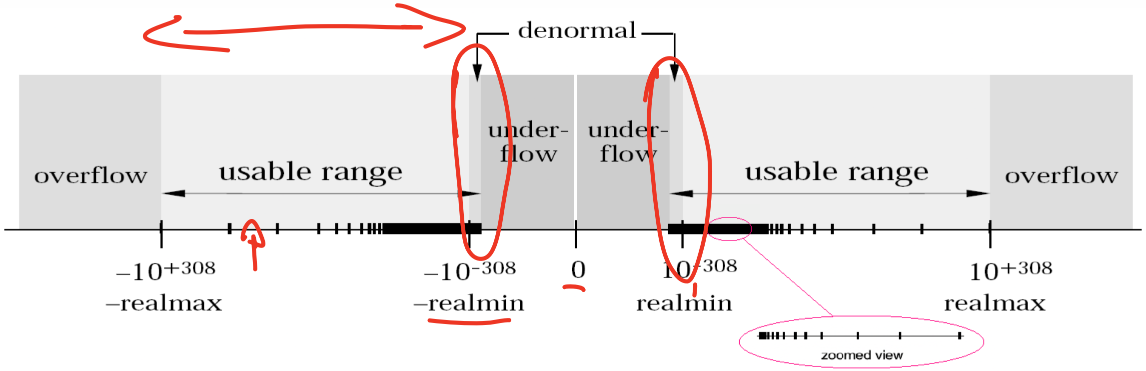
   Wichtig für die Darstellung von 0.

   Zahlen in der Nähe von 0

Doppelte Genauigkeit

Der denormalisierte Bereich ist im

Vergleich zum normalisierten sehr klein.



IEEE 754 Erläutert

Im folgenden Bild werden die Wertebereiche von Gleitkommazahlen auf der Zahlengeraden verdeutlicht. An deren Enden sind Werte für ± ∞ sowie für sog. "Nichtzahlen" (NaNs) reserviert.

3a. Infinity

s	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3b. NaN

s	1	1	1	1	1	1	1	1	≠ 0

IEEE 754 Erläutert

Im folgenden Bild werden die Wertebereiche von Gleitkommazahlen auf der Zahlengeraden verdeutlicht. An deren Enden sind Werte für ± ∞ sowie für sog. "Nichtzahlen" (NaNs) reserviert.

3a. Infinity

s	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3b. NaN

s	1	1	1	1	1	1	1	1	≠ 0

Für die Berechnung von Gleitkommazahlen in Mikrorechnern warden entweder viele Einzeloperationen (s.o.) benötigt, oder es werden spezielle (aufwendige) Rechenwerke implementiert.

Darstellung von reellen Zahlen

Eigenschaft	Einfache Genauigkeit	Doppelte Genauigkeit	Erw. Genauigkeit
Datenformat	32 Bit	64 Bit	80 Bit
Signifikand	24 Bit	53 Bit	64 Bit2
Größter relativer Fehler	2^{-24}	2^{-53}	2^{-64}
Genauigkeit	$\approx 1.3 \cdot 10^{-7}$	$\approx 2.2 \cdot 10^{-16}$	$\approx 1.1 \cdot 10^{-19}$
Biased Exponent E	8 Bit	11 Bit	15 Bit
Maximalwert Emax	255	2047	32767
Bias E0	127	1023	16383
Bereich für e	-126 bis +127	-1022 bis +1023	-16382 bis +16383
Kleinste positive Zahl	$2^{-126} \approx 1{,}2 \cdot 10^{-38}$	$2^{-1022} \approx 2{,}2 \cdot 10^{-308}$	$2^{-16382} \approx 2{,}2 \cdot 10^{-4932}$
Größte positive Zahl	$(2-2^{-23}) \cdot 2^{127}$	$(2-2^{-52}) \cdot 2^{1023}$	$(2-2^{-63}) \cdot 2^{16383}$
	$\approx 3{,}4 \cdot 10^{38}$	$\approx 1{,}8 \cdot 10^{308}$	$\approx 1{,}2 \cdot 10^{4932}$
Zahlenbereich	**Wert**	**Biased Exponent E**	**Signifikand s=g,b**
null (Einzelwert)	(-1)" 0	0	g=0,b=0
unnormiert	$(-1)^v \cdot 0.b \cdot 2^{1-E_0}$	0	g=0,b>0
normiert	$(-1)^v \cdot 1.b \cdot 2^{E-E_0}$	$0 < E < E_{max}$	g=1,b≥0
unendlich (Einzelwert)	$(-1)^v \cdot \infty$	E_{max}	g=1,b=0
Nichtzahl	NaN	E_{max}	g=1,b>0

$(-25)_{10} \rightarrow$

$(25)_{10} \rightarrow$

$25 \div 2 = 12 \quad R1$

$12 \div 2 = 6 \quad R0$

$6 \div 2 = 3 \quad R0$

$3 \div 2 = 1 \quad R1$

$1 \div 2 = 0 \quad R1$

$(25)_{10} \rightarrow 0b11001 \rightarrow 5b$

falsch (5b) EK $\rightarrow$ ~~0b00110~~

~~+1 1~~

~~0b00111~~

8bit: 0b 0001 1001 $(+25)_{10}$

EK 0b 1110 0110

+1 1

0b 1110 0111 ZK 8bit

$(-25)_{10}$

8 bit: 0b 1110 0111 => positive ganzzahl

$(231)_{10}$

$$ZK = -2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0$$

$$= (-25)_{10}$$

References

Cua Hock-Chuan. A Tutorial on Data Representation Integers, Floating-point Numbers, and Characters. Nanyang Technical University. Accessed April 2023.

Mariana Silva. Floating Point Representation. Illinois University. Accessed April 2023.

Steven Petryk. HOW TO: Adding IEEE-754 Floating point numbers. Accessed April 2023.

Sarah Harris. Digital Technik. TU Darmstadt. Accessed April 2023.

Bernhard Rinner. Entwurf digitaler Schaltungen. Accessed April 2023.