



Technische
Universität
Braunschweig



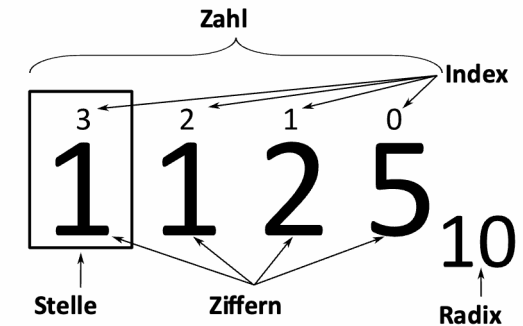
Informatik für Ingenieure – VL 2

Prof. Dr. Andres Gomez

Zusammenfassung der letzten Sitzung

2.2

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



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

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

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

1's column
2's column
4's column
8'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	$[0, 2^N - 1]^*$

*Bei der Interpretation als Ganzzahl ohne Vorzeichen

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

2.4

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

Art 2: Durch immer größer werdende Zweierpotenzen dividieren

Die heutigen Ziele

2.5

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 ...

2.6

Bits (Einheit *b*)

Höchstwertiges Bit (*msb*)

Niedrigstwertiges Bit (*lsb*)

10100010
└─┘ └─┘
most least
significant bit significant bit

Bytes (Einheit *B*) & Nibbles

byte
┌───────────┐
10010110
└───┘
nibble

Bytes

Höchstwertiges Byte (*MSB*)

Niedrigstwertiges Byte (*LSB*)

word
┌──────────┐
CE BF 9A D7
└─┘ └─┘
most least
significant byte significant byte

Doppelwort: 64 Bits

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	{		}	~	

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

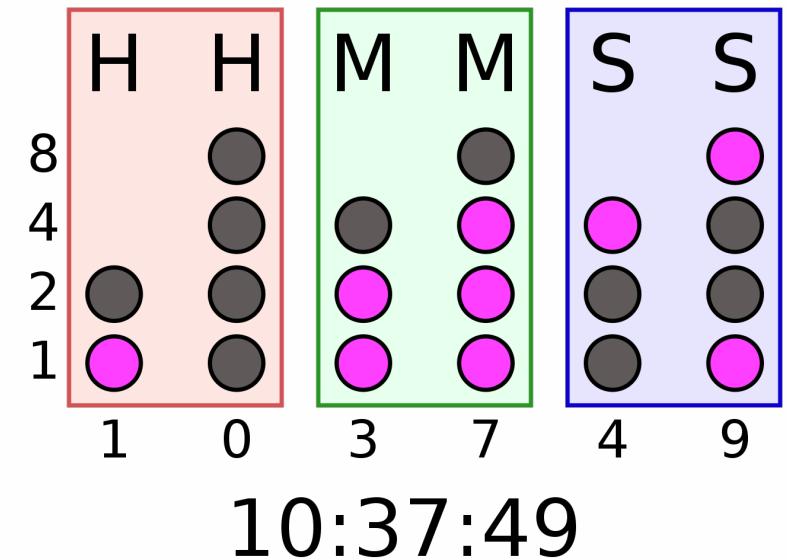
2.8

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

2.9

$10^3 = 1$ 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

2.10

Was ist der Wert von 2^{24} ?

Wie viele verschiedene Werte kann eine 32b Variable annehmen?

Addieren wir zwei 4-stellige Zahlen:

Dezimal:

$$\begin{array}{r} 3734 \\ + 5168 \\ \hline \end{array}$$

Binär:

$$\begin{array}{r} 0b1001 \\ + 0b0111 \\ \hline \end{array}$$

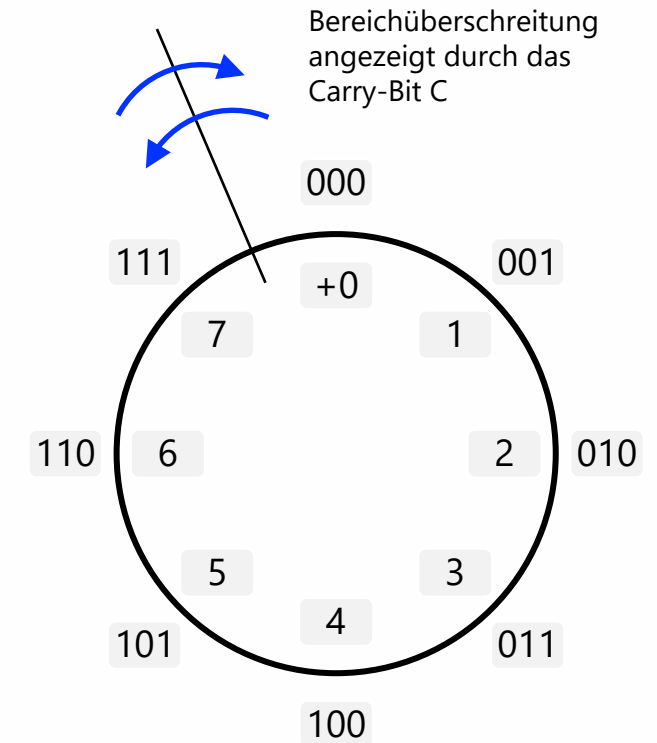
Überlauf

2.11

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

2.12

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

Vorzeichenbit ist höchstwertiges Bit (msb)

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

Positive Zahl: Vorzeichenbit = 0

Negative Zahl: Vorzeichenbit = 1

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

Beispiel:

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

$$+6 = 0b0110$$

$$-6 = 0b1110$$

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

Darstellung als Vorzeichen/Betrag: Probleme

2.13

Addition schlägt **fehl**

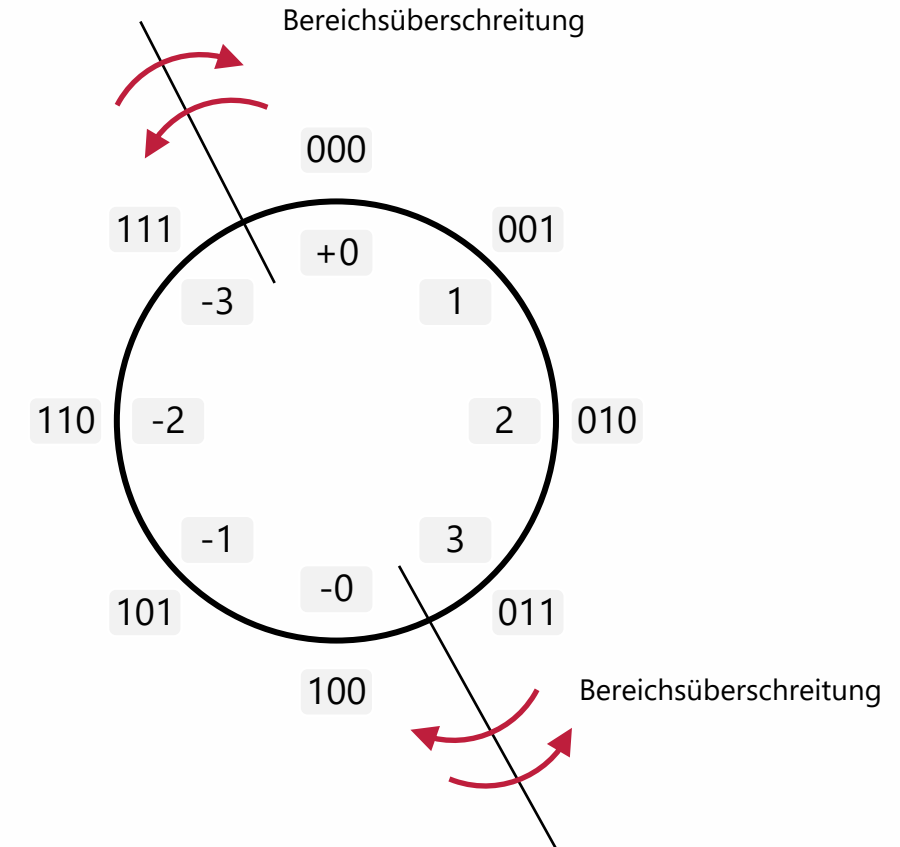
Beispiel: $-3 + 3$:

$$\begin{array}{r} 0b111 \\ + 0b011 \\ \hline \ddot{U}=1 \quad 0b010 \quad (\text{falsch!}) \end{array}$$

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

2.15

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

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

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

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

$$\begin{array}{l} +5 \text{ in EK: } 0b0101 \\ -5 \text{ in EK: } 0b1010 \end{array}$$

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

$$\begin{array}{l} +0 \text{ in EK: } 0b0000 \\ -0 \text{ in EK: } 0b1111 \end{array}$$

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):

$$\begin{array}{l} \text{Eindeutig: } \text{ZK}(+0) = \text{ZK}(-0) \\ = 0b0000 \end{array}$$

$$\begin{array}{r} 0b1010 \quad \text{EK} \\ +0b0001 \quad 1 \\ \hline 0b1011 \quad \text{ZK} \end{array}$$

$$\begin{array}{l} +5 \text{ in ZK: } 0b0101 \\ -5 \text{ in ZK: } 0b1011 \end{array}$$

Zahlendarstellung im Zweierkomplement

2.16

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 $\rightarrow b_{n-1} = 1$

positive $\rightarrow b_{n-1} = 0$

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

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

Zweierkomplement arithmetisch bilden

2.17

Algorithmus:

Alle Bits invertieren ($0 \rightarrow 1$, $1 \rightarrow 0$)

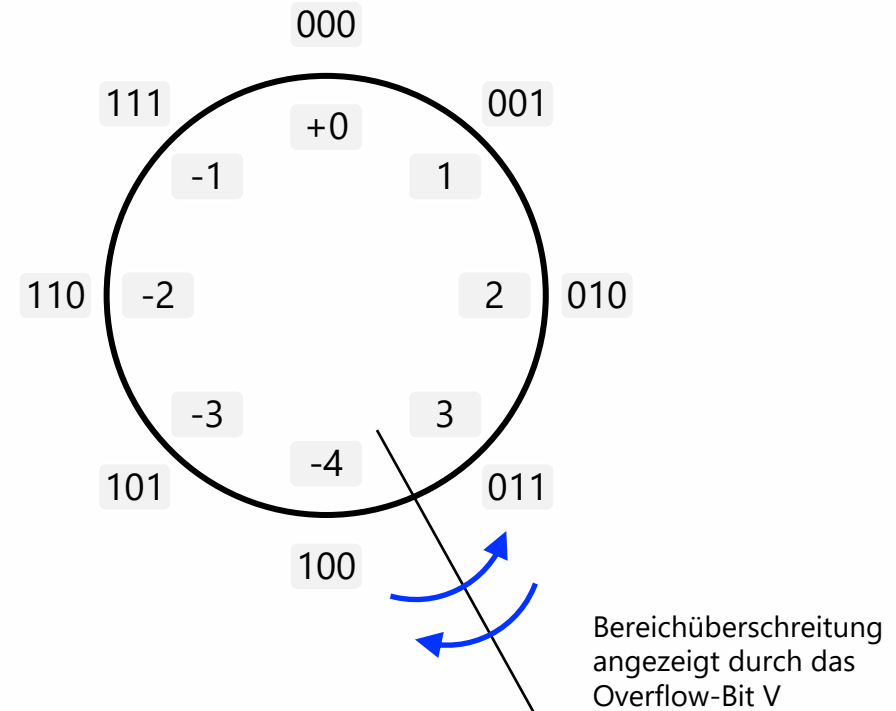
Dann 1 addieren

In **beide** Richtungen anwendbar

Vorzeichenwechsel: $k \rightarrow -k$

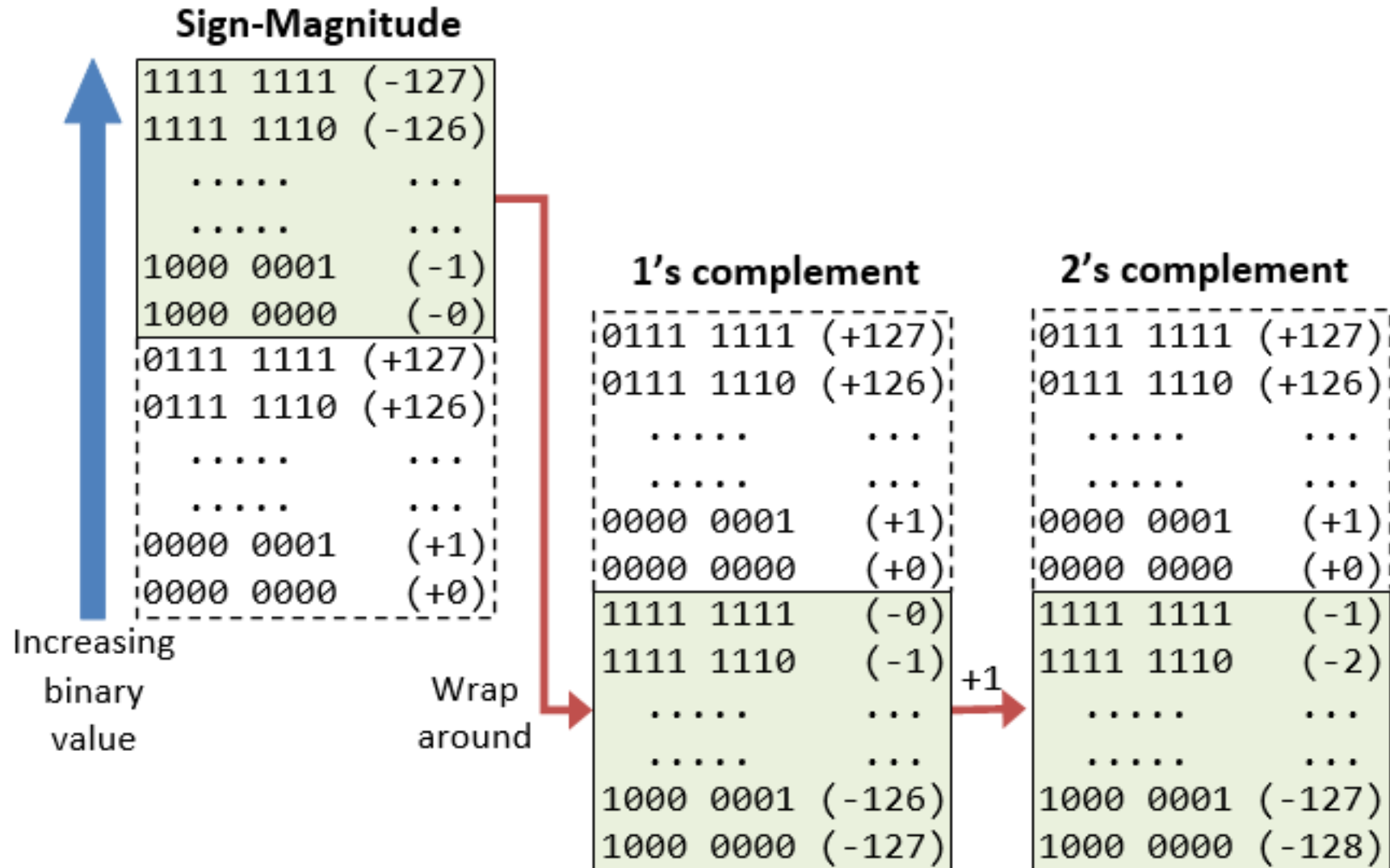
Beispiel: Vorzeichenwechsel von $(3)_{10} = 0b00011$

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



Visualisierung verschiedener Darstellungen

2.18



Zweierkomplement

2.20

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

Fallunterscheidung bei Addition im 2er Komplement:

$$\begin{array}{rcll} \text{Fall 1:} & 0b0000 & 0001 & \equiv (+1)_{10} \\ & + & 0b1111 & 1111 \equiv (-1)_{10} \\ \ddot{U}=1 & \hline & 0b0000 & 0000 \equiv (0)_{10} \quad \checkmark \end{array}$$

$$\begin{array}{rcll} \text{Fall 2:} & 0b1111 & 1111 & \equiv (-1)_{10} \\ & + & 0b1111 & 1110 \equiv (-2)_{10} \\ \ddot{U}=1 & \hline & 0b1111 & 1101 \equiv (-3)_{10} \quad \checkmark \end{array}$$

$$\begin{array}{rcll} \text{Fall 3:} & 0b0000 & 0001 & \equiv (+1)_{10} \\ & + & 0b0000 & 0010 \equiv (+2)_{10} \\ \ddot{U}=0 & \hline & 0b0000 & 0011 \equiv (+3)_{10} \quad \checkmark \end{array}$$

Zweierkomplement

2.21

Fallunterscheidung bei Addition im 2er Komplement:

Fall 4:

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

Fall 5:

$$\begin{array}{rcll} & 0b1000 & 0000 & \equiv (-128)_{10} \\ + & 0b1111 & 1111 & \equiv (-1)_{10} \\ \hline \ddot{U}=1 & 0b0111 & 1111 & \equiv (+127)_{10} \end{array} \quad \text{Falsch}$$

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

2.22

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 schmalen 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

2.23

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 :

Erweitern durch Auffüllen mit Nullbits

2.24

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:

Noch ein letztes Wort zum Thema Umwandlung

2.25

Konvertierung zwischen vorzeichenbehafteten \leftrightarrow 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

2.26

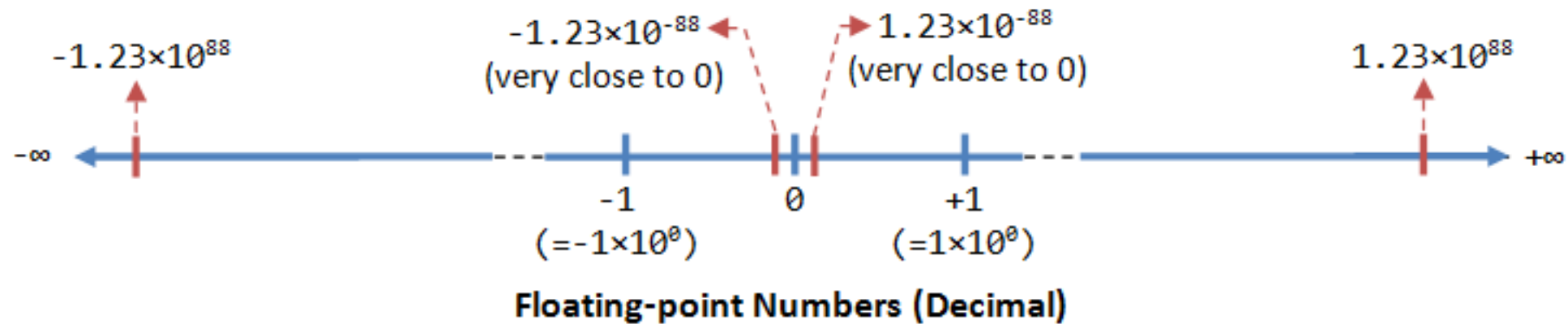
Wortbreite	Vorzeichenlos	Vorzeichen/Betrag	Zweierkomplement
8 bit	0 ... 255	-127 ... -0 , +0 ... +127	-128 ... 0 ... +127
16 bit	0 ... 65535	-32767 ... -0 , +0 ... +32767	-32768 ... 0 ... +32267
32 bit	0 ... 4Gi - 1	-(2Gi-1) ... -0 , +0 ... +(2Gi-1)	-2Gi ... 0 ... +2Gi-1
n bit	0 ... $2^n - 1$	$-(2^{n-1}-1)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

Darstellung von reellen Zahlen

2.27

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)

2.28

Vorgegebene Anzahl von Bits vor
und nach dem Komma:

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

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

Beispiel:

$$0b1100.0011 = 2^3 + 2^2 + 2^{-3} + 2^{-4} = 12.1875$$

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)

2.29


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: $V \cdot S \cdot B^E$

Festkomma: 0b10110010.001


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 2^E fest.

Mathematische Operationen mit Gleitkomma

2.30

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:

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

IEEE 754 Gleitkommadarstellungsformat

2.31

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

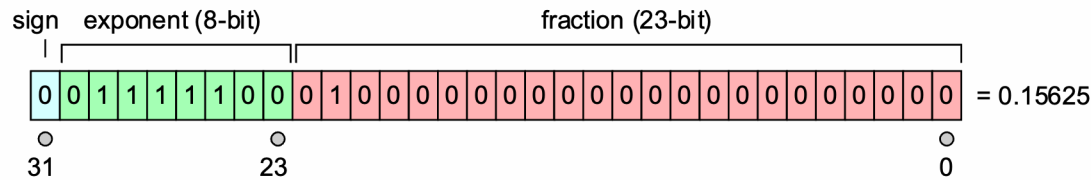
Für $E \geq 1$: $g = 1$ (normalisiert)

Exponent: $e = E - E_0$

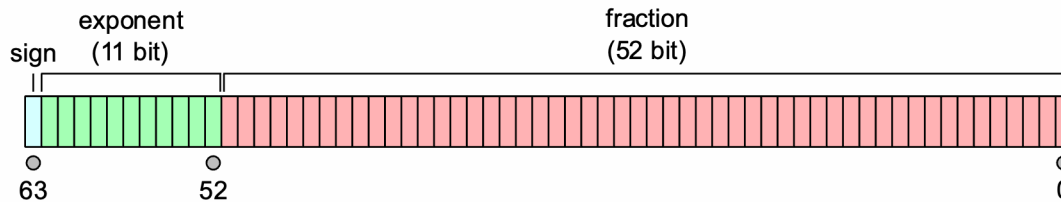
Exzesscode, mit E_0 abhängig
von der Präzision

Für $E = 0$: $g = 0$ (denormalisiert)

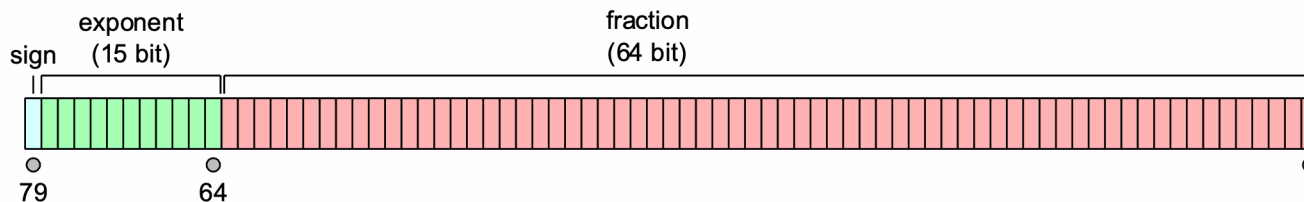
Einfache
Genauigkeit



Doppelte
Genauigkeit



Erweiterte
Genauigkeit



IEEE 754 Erläutert

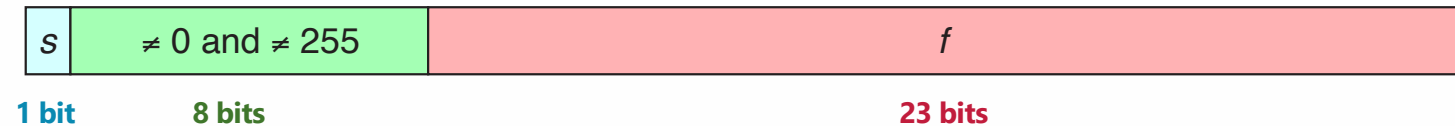
2.32

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.

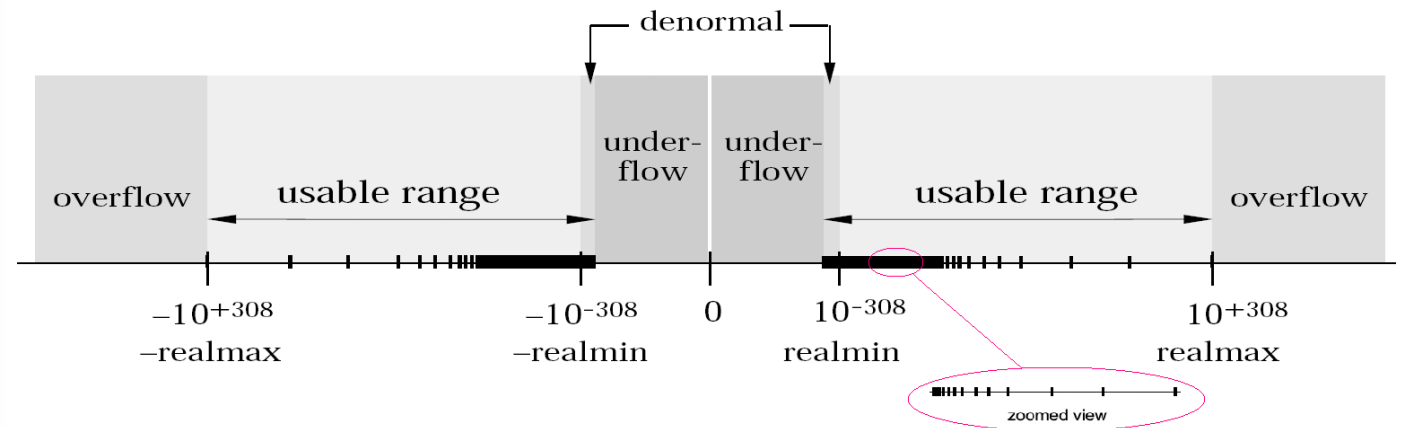
Zahlen in der Nähe von 0



Doppelte Genauigkeit

Der denormalisierte Bereich ist im

Vergleich zum normalisierten sehr klein.

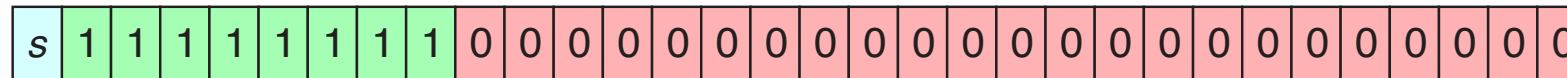


IEEE 754 Erläutert

2.33

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

3a. Infinity



3b. NaN



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

Darstellung von reellen Zahlen

2.34

Eigenschaft	Einfache Genauigkeit	Doppelte Genauigkeit	Erw. Genauigkeit
Datenformat	32 Bit	64 Bit	80 Bit
Signifikand	24 Bit	53 Bit	64 Bit ²
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 E _{max}	255	2047	32767
Bias E ₀	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)	$(-1)^v \cdot 0$	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