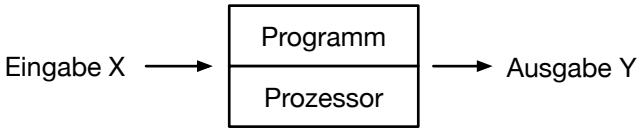
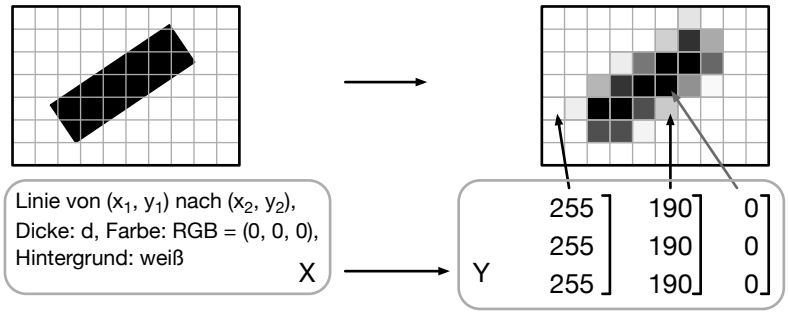
2 Darstellung von Zahlen und Zeichen

Computer- bzw. Prozessorsysteme führen Transformationen durch, die Eingaben X auf Ausgaben Y abbilden, d.h. $Y = f(X)$.

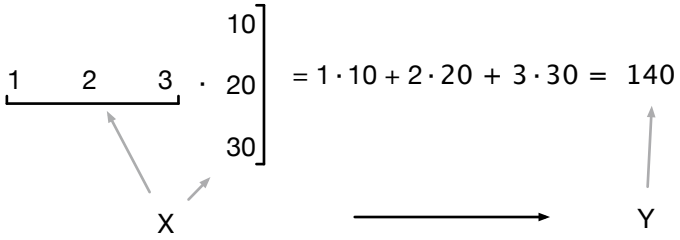


Die Art und Weise, wie diese Transformationen durchgeführt werden, ist durch die Programme festgelegt, die von einem Prozessor ausgeführt werden. Beispiele:

- Dokument drucken:
  - X: Dokument bzw. Datensatz in einer Applikation
  - Y: Befehle/Daten, die an den Drucker geschickt werden müssen, damit dieser das (durch X repräsentierte) Dokument druckt
  - Programm: Applikation, aus der heraus das Dokument gedruckt wird (z.B. Textverarbeitungsprogramm) sowie der Druckertreiber
- Rastern von Grafiken: X = Repräsentation eines Objekts (z.B. Linie);
  Y = Farbintensitätswerte von Pixeln



- Berechnungen: Y aus X berechnen; z.B. X = zwei Vektoren, Y = Skalarprodukt



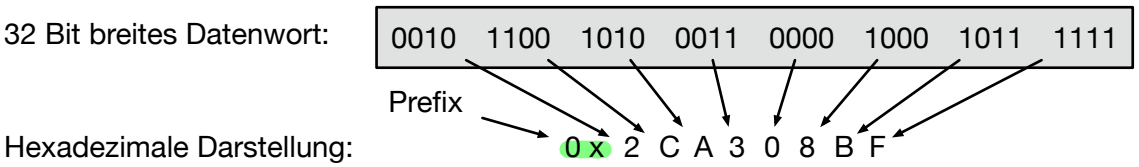
X und Y sind Daten, die als *Zahlen* oder als *Zeichen* interpretiert werden können. Sie werden in Computersystemen durch sog. *Bits* repräsentiert.

2.1 Bits, Byte, Datenworte und Logikpegel

Daten werden in Computersystemen durch Bits dargestellt bzw. als Bits verarbeitet. Der Begriff *Bit* steht für *binary digit* und meint *Binärziffer*, d.h. Ziffern, die nur Werte 0 und 1 annehmen können. Bei der Verarbeitung von Daten durch elektrische Schaltungen entspricht ‘‘0’’ oft dem sog. *Low*-Pegel, z.B. -0,3 ... +1,3 Volt, und ‘‘1’’ dem sog. *High*-Pegel, z.B. +2,3 ... +5,3 Volt.

Darüber hinaus findet man auch andere Zuordnungen/Spannungsbereiche. Bei der seriellen Schnittstelle RS-232 beispielsweise entsprechen Spannungen zwischen +3 V ... +15 V dem Low-Pegel, während Spannungen zwischen -15 V ... -3 V High-Pegel darstellen.

Mit einem einzelnen Bit können nur zwei Zustände, High und Low, dargestellt werden. Um mehr als zwei Zustände gleichzeitig abzubilden, werden mehrere Bits zu einem *Datenwort* zusammengefasst. Mit einem Datenwort der Breite n Bits lassen sich 2^n verschiedene Low-/High-Kombinationen darstellen.
Nachfolgende Abbildung zeigt ein Datenwort der Breite $n = 32$ Bit sowie die entsprechende Darstellung in Hexadezimal-Schreibweise.

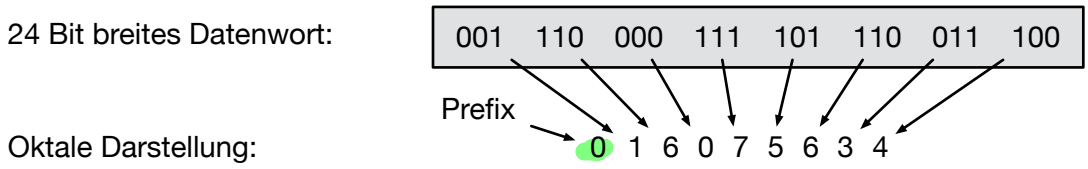


Die hexadezimale Darstellung wird häufig verwendet, da hier immer vier Bits (sog. *Nibble*) zu einer einzelnen Ziffer zusammengefasst werden:

0: 0000	1: 0001	2: 0010	3: 0011	4: 0010	5: 0101	6: 0110	7: 0111
8: 1000	9: 1001	A: 1010	B: 1011	C: 1100	D: 1101	E: 1110	F: 1111

So lassen sich auch längere binäre Datenworte ohne großen Platzbedarf darstellen. Gleichzeitig kann durch die feste 4-zu-1-Abbildung der Wert der einzelnen Bits direkt extrahiert werden.
Zur Kennzeichnung einer hexadezimalen Codierung wird das Prefix ‘‘0x’’ verwendet, d.h. hexadezimal codierten Zahlen wird ‘‘0x’’ vorangestellt.

Seltener findet man oktale Codierungen. Hier wird das Prefix ‘‘0’’ verwendet. Bei oktaler Codierung werden immer 3 Bits zu einer Ziffer zusammengefasst.



0: 000 1: 001 2: 010 3: 011 4: 010 5: 101 6: 110 7: 111

In Computersystemen werden häufig Worte der Breite 8, 16, 32 oder 64 Bit verwendet. Datenworte mit der Wortbreite 8 Bit werden *Byte* genannt. Ein Byte wird dabei oft als elementare Datenwortgröße angesehen. Alle anderen Datenworte sind dann ein ganzzahliges Vielfaches eines Bytes.

Nachfolgende Abschnitte zeigen, wie in Computersystemen mit solchen binären Datenworten Zahlen und Zeichen dargestellt werden. Die darauf folgenden Kapitel zeigen, wie diese Datenworte/Zahlen/Zeichen von Prozessoren verarbeitet werden.

2.2 Zeichen

Zeichen sind Symbole (z.B. ‘a’, ‘b’, ‘c’, ...), mit deren Hilfe Dinge beschrieben werden können. Zur Darstellung von Texten werden Zeichen zu Zeichenketten (Worte) kombiniert und Zeichenketten in Anordnungen (Sätze) gruppiert. Die ‘‘Beschreibung’’ findet dadurch statt, dass unser Gehirn beim Lesen lernen die Bedeutung der verschiedenen Zeichenketten (Symbol-Kombinationen) sowie die Bedeutung verschiedener Anordnungen gelernt hat. In Computersystemen werden Zeichen durch Bits repräsentiert. Nachfolgende Tabelle zeigt die Codierung von Zeichen gemäß ASCII-Standard.

	0x0…	0x1…	0x2…	0x3…	0x4…	0x5…	0x6…	0x7…
…0	NUL	DLE	SP	0	@	P	`	p
…1	SOH	DC1	!	1	A	Q	a	q
…2	STX	DC2	"	2	B	R	b	r
…3	ETX	DC3	#	3	C	S	c	s
…4	EOT	DC4	$	4	D	T	d	t
…5	ENQ	NAK	%	5	E	U	e	u
…6	ACK	SYN	&	6	F	V	f	v
…7	BEL	ETB	'	7	G	W	g	w
…8	BS	CAN	(	8	H	X	h	x
…9	HT	EM	)	9	I	Y	i	y
…A	NL	SUB	*	:	J	Z	j	z
…B	VT	ESC	+	;	K	[	k	{
…C	NP	FS	,	<	L	\	l	|
…D	CR	GS	-	=	M	]	m	}
…E	SO	RS	.	>	N	^	n	~
…F	SI	US	/	?	O	_	o	DEL

'A' = 0x41

nicht druckbare Steuerzeichen

Druckbare Zeichen

‘‘ASCII’’ (oft auch US-ASCII) steht für *American Standards Code for Information Interchange* und ist ein weit verbreiteter Standard zur Codierung von 128 ausgewählten Zeichen durch 7 Bit breite Datenworte.
Druckbare Zeichen, d.h. Zeichen, die auch am Bildschirm/Drucker ausgegeben werden können, befinden sich ab Bitkombination 0x20, d.h. Zeichen 33 - 128.
Die unteren 32 Zeichen, d.h. Bitkombinationen 0x00, 0x01, ... , 0x1F definieren sog. *Steuerzeichen*. Steuerzeichen wurden früher dafür verwendet um Fernschreiber anzusteuern.

0x00 (NUL): Null	0x10 (DLE): Data link escape
0x01 (SOH): Start of header	0x11 (DC1): Device control 1
0x02 (STX): Start of text	0x12 (DC 2): Device control 2
0x03 (ETX): End of text	0x13 (DC 3): Device control 3
0x04 (EOT): End of transmission	0x14 (DC 4): Device control 4
0x05 (ENQ): Enquiry	0x15 (NAK): Negative acknowledge
0x06 (ACK): Acknowledge	0x16 (SYN): Synchronous idle
0x07 (BEL): Bell	0x17 (ETB): End of transmission block
0x08 (BS): Backspace	0x18 (CAN): Cancel
0x09 (HT): Horizontal tab	0x19 (EM): End of medium
0x0A (LF): Line feed; new line	0x1A (SUB): Substitute
0x0B (VT): Vertical tab	0x1B (ESC): Escape
0x0C (FF): Form feed; new page	0x1C (FS): File separator
0x0D (CR): Carriage return	0x1D (GS): Group separator
0x0E (SO): Shift out	0x1E (RS): Record separator
0x0F (SI): Shift in	0x1F (US): Unit separator

Die meisten Steuerzeichen werden heute nur noch selten verwendet. Häufig verwendet wird beispielsweise 0x00 wird, um das Ende von Zeichenketten anzuzeigen, 0x0A um einen Zeilenumbruch zu markieren, 0x09 für Tabulatoren.

Der ASCII-Code definiert ausschließlich die Codierung der *in Amerika* häufig verwendeten Zeichen. Codierungen für international verwendete Zeichen wie bspw. deutsche Umlaute ‘‘ä’’, ‘‘ö’’ und ‘‘ü’’ sowie ‘‘ß’’ etc. werden nicht definiert. Dazu muss der ASCII-Zeichensatz erweitert werden. Beispiele hierzu sind der Standard ISO 8859-1 (Latin-1) oder Zeichentabellen, wie sie unter MS-DOS eingesetzt wurden (z.B. Codepage 850 für Westeuropa).

Heute wird häufig der Unicode-Zeichensatz verwendet. Dieser hat zum Ziel, jedem auf der Welt verwendeten Schriftzeichen eine eindeutige Zahl zuzuweisen.

Zur Codierung dieser Zahlen werden häufig UTF-8 und UTF-16 eingesetzt. Diese Verfahren codieren den Unicode-Zeichensatz in *variable* Wortbreiten. So können zur Codierung häufig vorkommender Zeichen geringere Wortbreiten verwendet werden als zur Codierung seltener vorkommender Zeichen. Diese Form der Komprimierung sorgt dafür, das Text aus Sprachen, die auf dem lateinischen Alphabet basieren, effizient abgespeichert bzw. über das Internet übertrag werden können.

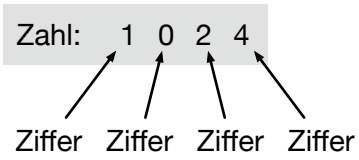
Nachfolgende Abbildung zeigt die Codierung gemäß UTF-8.

Codierung	**Unicode-Zeichen**
0xxxxxxx	0x00 - 0x7F (entspricht ASCII)
110xxx 10xxxxxx	0x080 - 0x7FF
1110xxx 10xxxxxx 10xxxxxx	0x0800 - 0xFFFF
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	0x010000 - 0x10FFFF

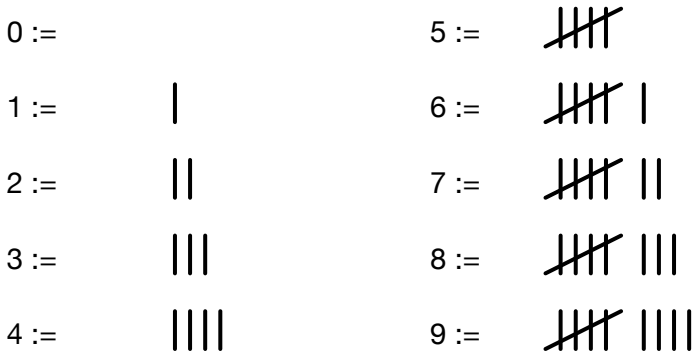
Im Gegensatz dazu wird in UTF-32 jedes Unicode-Zeichen mit 32 Bit codiert. Vorteil: Einfach zu codieren; Nachteil: Hoher Speicherbedarf für Texte.

2.3 Zahlen

Zahlen dienen zur Darstellung von Größen/Beträgen. Sie werden durch Ziffern dargestellt.



Ziffern sind Zeichen, die jedem Element einer Symbol-Menge (z.B. {‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’}) *ein Vielfaches eines Grundbetrags als Wert zuordnen*. Beispiel: ‘0’ ist ‘‘nichts’’ bzw. *keinmal* der Grundbetrag, ‘1’ *ist* der Grundbetrag, ‘2’ ist *zweimal* so viel wie der Grundbetrag; ‘3’ ist *dreimal* so viel wie der Grundbetrag, etc.



Die Menge der in einem Zahlensystem vorgesehenen Symbole wird *Basis b* genannt. Beispiel: Im Zahlensystem zur Basis b = 2 gibt es nur zwei Symbole: ‘0’ und ‘1’.
Mit *einer* Ziffer können nur *b* verschiedene Dinge/Werte dargestellt werden. Um mehr als *b* verschiedene Werte abzubilden werden mehrere Ziffern aneinandergereiht. Dabei erhöht sich mit jeder weiteren Ziffer die Anzahl unterschiedlicher Symbol-Kombinationen um den Faktor b. Durch Aneinanderreihung von *n* Ziffern zu einer n Stellen langen *Zahl* lassen sich $\underbrace{b \cdot b \cdot \ldots \cdot b}_{\text{n mal}} = b^n$ verschiedene Symbolkombinationen und damit b^n verschiedene Werte/Beträge darstellen.

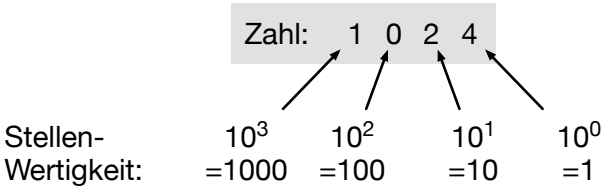
Nachfolgende Abbildung zeigt die Symbole zur Darstellung von Beträgen mit zwei Ziffern aus der Symbolmenge ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’.

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Abbildungsregel

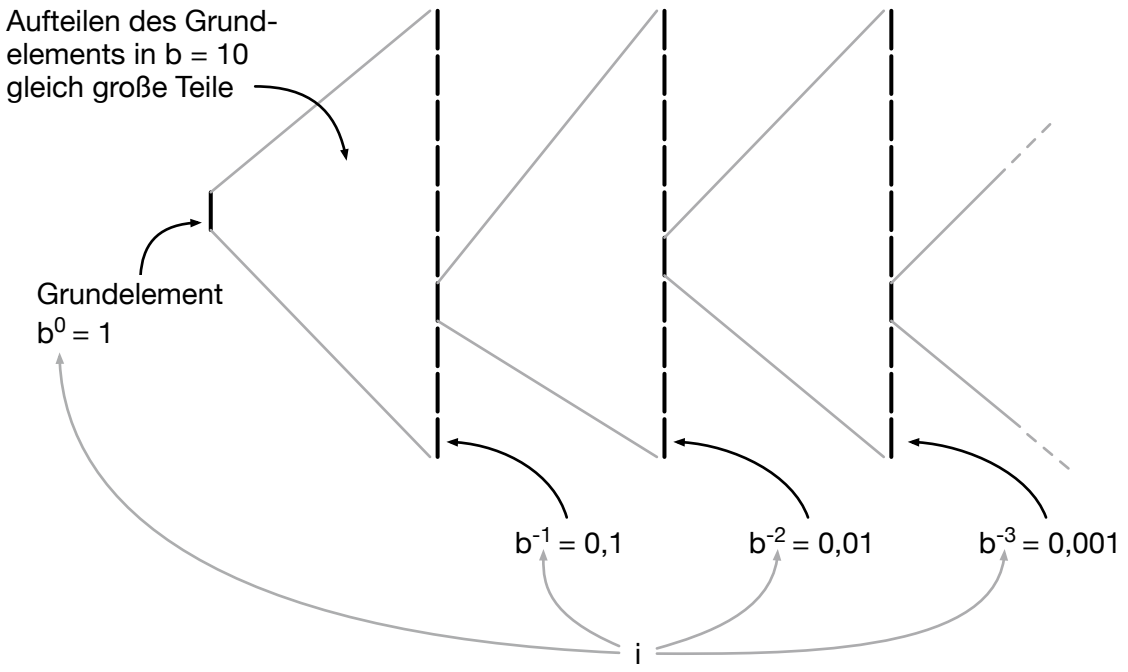
- Der kleinste Wert wird dadurch repräsentiert, dass alle Ziffern das Symbol des niedrigsten Werts darstellen.
- Ausgehend vom kleinsten Wert wird der nächst höhere Wert stets dadurch repräsentiert, dass bei der rechtesten Ziffer das dem nächst höheren Ziffern-Wert entsprechende Symbol ausgewählt wird.
- Ist bei einer Ziffer bereits das werthöchste Symbol ausgewählt, wird bei dieser Ziffer das wertniedrigste Symbol ausgewählt. Gleichzeitig wird die links angrenzende Ziffer durch das dem nächst höheren Ziffern-Wert entsprechende Symbol ersetzt.

Durch dieses Vorgehen haben die einzelnen Ziffern-Positionen unterschiedliche Wertigkeiten. Numeriert man die Ziffern-Positionen i von rechts nach links durch, beginnend mit $i = 0$, dann hat jede Ziffernposition den Wert b^i. Beispiel mit $b = 10$:

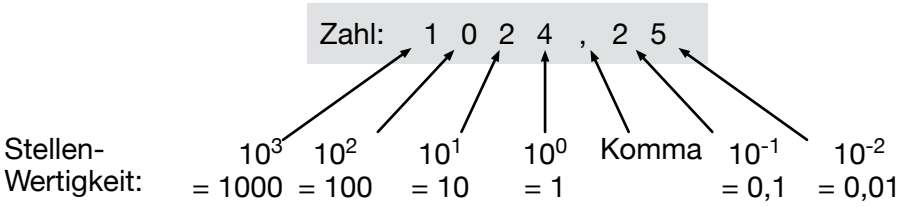


Der Wert der Zahl ergibt sich zu $1 \cdot 1000 + 0 \cdot 100 + 2 \cdot 10 + 4 \cdot 1 = 1024$.

Im Gegensatz zu Ziffern-Positionen links von $i = 0$ stellen Ziffern-Positionen rechts von $i = 0$, d.h. i < 0, nicht ein *Vielfaches* des Grundelements dar, sondern einen *Bruchteil* des Grundelements. Nachfolgende Abbildung zeigt am Beispiel $b = 10$, wie die Stellenwertigkeit von links nach rechts auf b^i, d.h. b^{-1}, b^{-2}, b^{-3}, ... reduziert wird.



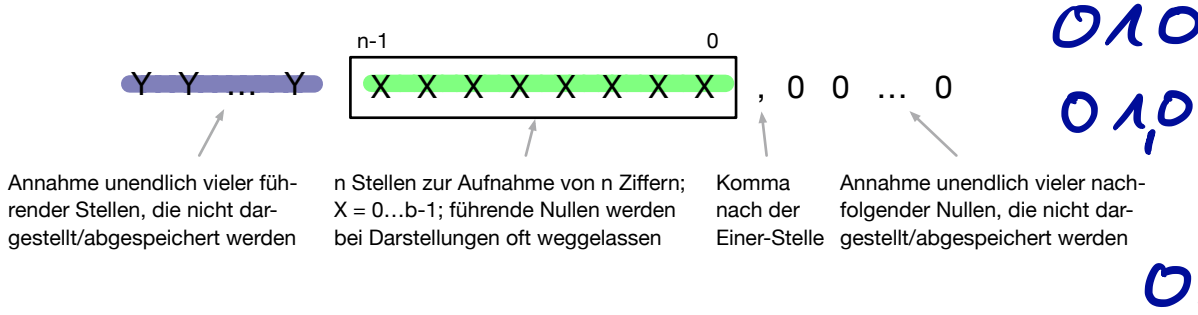
Sind Stellen i < 0 vorhanden, so wird der Übergang (i = 0) → (i < 0) durch das *Komma*-Symbol gekennzeichnet.



Da es unendlich viele Zahlen gibt, verfügen Zahlen (theoretisch) über unendlich viele Stellen vor bzw. nach dem Komma. Für in der Praxis auftretende Zahlen werden in der Regel jedoch nur wenige Stellen vor und wenige Stellen nach dem Komma benötigt. Die restlichen (unendliche vielen) führenden bzw. nachlaufenden Nullen werden nicht dargestellt.

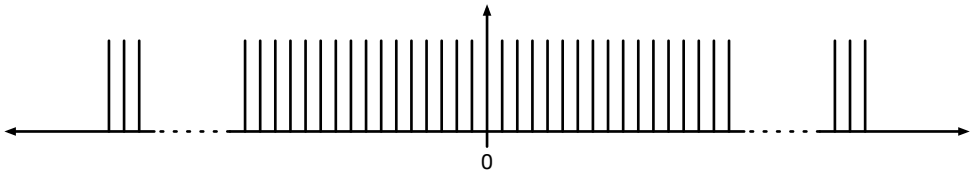
2.4 Codierung von Festkommazahlen

Festkommazahlen sind Zahlen, bei denen das Komma an einer zuvor vereinbarten, d.h. *festen* Position steht. Nachfolgende Abbildung zeigt eine solche Festkommazahl:



X steht für die Ziffern 0, 1, ... , b-1, wobei b die *Basis* des verwendeten Zahlensystems darstellt (z.B. b = 2 für Binärzahlen, b = 10 für Dezimalzahlen, ...).
n ist die *Wortbreite*, d.h. es stehen n Bits zum Abspeichern der Zahl zur Verfügung.
Y steht für die unendlich vielen Stellen, die nicht mit abgespeichert werden.

Festkommazahlen funktionieren nach dem zuvor beschriebenen Prinzip ‘‘Vielfaches eines Grundelements’’. Aus diesem Grund sind die Abstände zwischen zwei benachbarten Zahlen stets gleich groß (Äquidistanz).



Vorzeichenlose Festkommazahlen

Vorzeichenlose Festkommazahlen haben kein Vorzeichen, d.h. sie sind stets positiv. Der Wert v (v = value) einer vorzeichenlosen Festkommazahl ergibt sich zu:

$$v = (a_{n-1} \cdot b^{n-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0) \cdot b^r$$

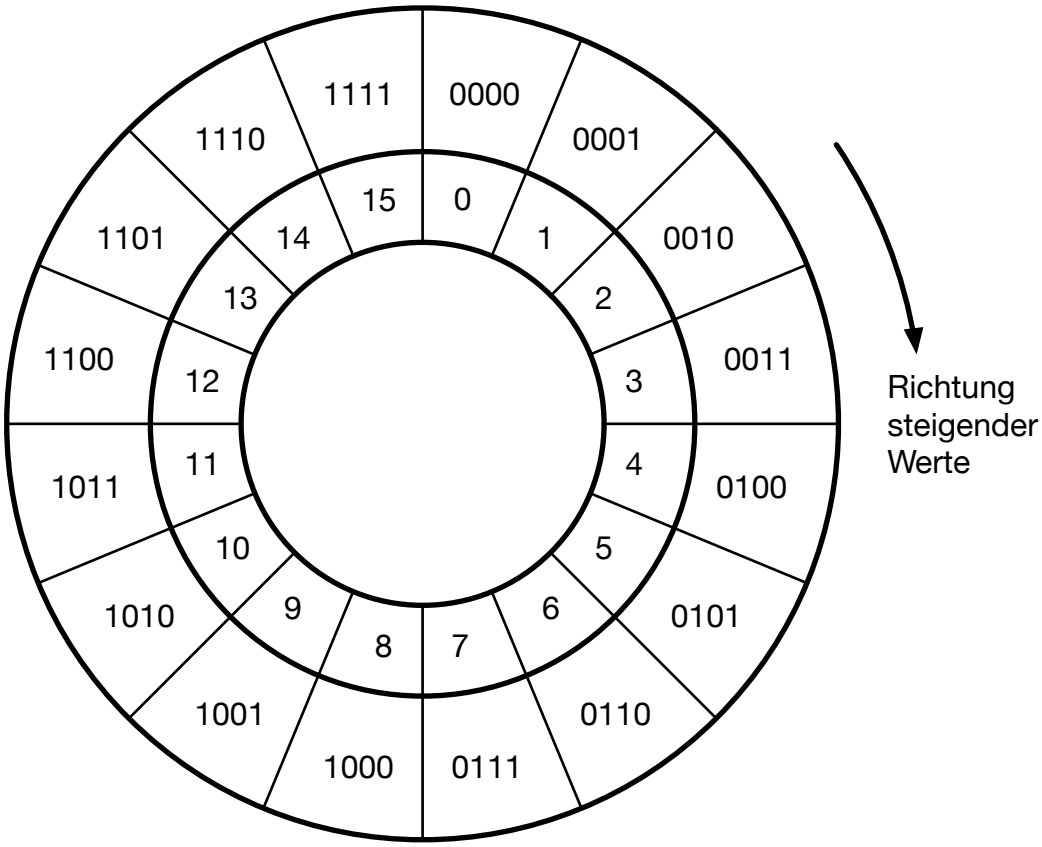
- n ist die Stellenzahl, d.h. die maximale Menge an Ziffern, die zur Darstellung bzw. Abspeicherung der Zahl vorgesehen ist. In Prozessoren wird häufig eine Stellenzahl von n = 8, 16, 32 oder 64 (Binär-) Stellen verwendet.
  In der Mathematik gibt es keine begrenzte Stellenzahl; dort gilt $n \to \infty$.
- b ist die Basis des Zahlensystems, z.B. 10 für das Dezimalsystem (Ziffern $0 \ldots 9$) oder 2 für Binärzahlen (Ziffern 0 und 1). Ziffern an der Stelle i haben die Wertigkeit b^i. In Prozessoren wird aufgrund der Darstellung von Werten durch

Logik-Pegel ‘‘Low’’ und ‘‘High’’ als Basis $b = 2$ verwendet.

- Die Koffizienten a_i sind die Ziffern an den Stellen i. Die Werte der Ziffern liegen im Bereich $0...(b-1)$ und geben an, wie oft die Wertigkeit der jeweiligen Stelle zum Wert der Zahl beiträgt.
- Der Wert von r (r = radix) legt die Position des Kommas fest:
  - $r = 0$: Dieser Fall ist der Normalfall: Durch Multiplikation mit $b^r = b^0 = 1$ bleibt $v = a_{n-1} \cdot b^{n-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0$. Das Komma steht hinter der Einer-Stelle und wird weggelassen. Es werden ganze Zahlen mit den Werten $0, 1, \ldots, b^n - 1$ dargestellt.
  - $r > 0$: Durch Multiplikation mit b^r können größere Zahlen dargestellt werden, jedoch auf Kosten geringerer Genauigkeit. Die Ziffern der Zahl werden um r Stellen *nach links* geschoben, die frei werdenden Positionen werden mit Nullen aufgefüllt. Das Komma wird weggelassen. Darstellungsbeispiel einer Festkommazahl für $n = 8$ und $r = 3$: xxxxxxxx000. Die Zeichen ‘‘x’’ stehen dabei jeweils für eine der Ziffern $a_{n-1} \ldots a_0$.
  - $r < 0$: Da $r < 0$, entspricht die Multiplikation mit b^r einer Division durch $b^{|r|}$, d.h. das (nach der Einer-Stelle implizit stehende) Komma wird um r Stellen nach links geschoben. Die Genauigkeit erhöht sich auf Kosten der größtmöglich darstellbaren Zahl. Darstellungsbeispiel für $n = 8$ und $r = -3$: xxxxx,xxx.

Im folgenden werden nur noch Dezimalzahlen ($b = 10$) und Binärzahlen ($b = 2$) betrachtet.

Nachfolgender Zahlenring zeigt die Zuordnung von Binär- zu Dezimalzahlen für diese Codierung:



Die Darstellung zeigt, dass die Richtung steigender Werte bei beiden Codierungen (Binär und Dezimal) identisch ist. Als Folge können bei dieser Darstellung für die gewählte Binärcodierung dieselben Rechenregeln angewendet werden, wie bei Dezimalzahlen. Beispiel:

- $2_{10} + 1_{10} = 3_{10}$
- $0010_2 + 0001_2 = 0011_2$

Vorzeichenbehaftete Festkommazahlen

Es gibt verschiedene Möglichkeiten, binäre vorzeichenbehaftete Festkommazahlen darzustellen:

- Vorzeichen und Betrag
- Einer-Komplement
- Zweier-Komplement

Vorzeichen und Betrag

Bei dieser Darstellung werden Vorzeichen und Betrag der Zahl separat abgespeichert:

- Das Vorzeichen wird repräsentiert durch das höherwertigste Bit: Hat das Bit den Wert 0, ist die Zahl positiv, hat das Bit den Wert 1, ist die Zahl negativ.
- Der Betrag der Zahl wird durch die restlichen Bits dargestellt.

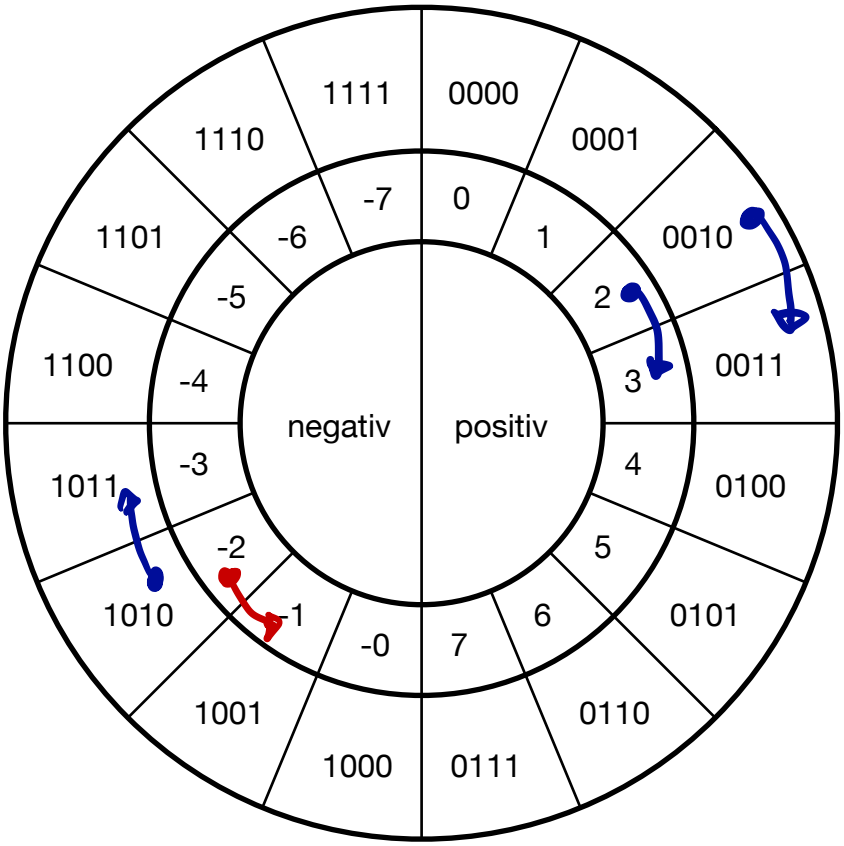
Ob eine Zahl positiv oder negativ ist, kann direkt am MSB abgelesen werden. Zur Negation einer Zahl muss nur das höherwertigste Bit geändert werden.

Ein Problem bei dieser Darstellung ist die doppelte Null:

- $00\ldots000_2 \Rightarrow +0$
- $10\ldots000_2 \Rightarrow -0$

Nachfolgende Abbildung zeigt für $n = 4$ die Zuodnung von Binär- zu Dezimalzahlen.

- Für positive Zahlen ist die Richtung steigender Werte für Binär- und Dezimalzahlen die selbe.
- Für negative Zahlen ist die Richtung jedoch unterschiedlich; Beispiel:
  - $1010_2 + 0001_2 = 1011_2$: Bewegung im Uhrzeigersinn
  - $-2_{10} + 1_{10} = -1_{10}$: Bewegung gegen den Uhrzeigersinn
  - Ergebnis falsch: $-1_{10} \neq 1011_2$



Aufgaben

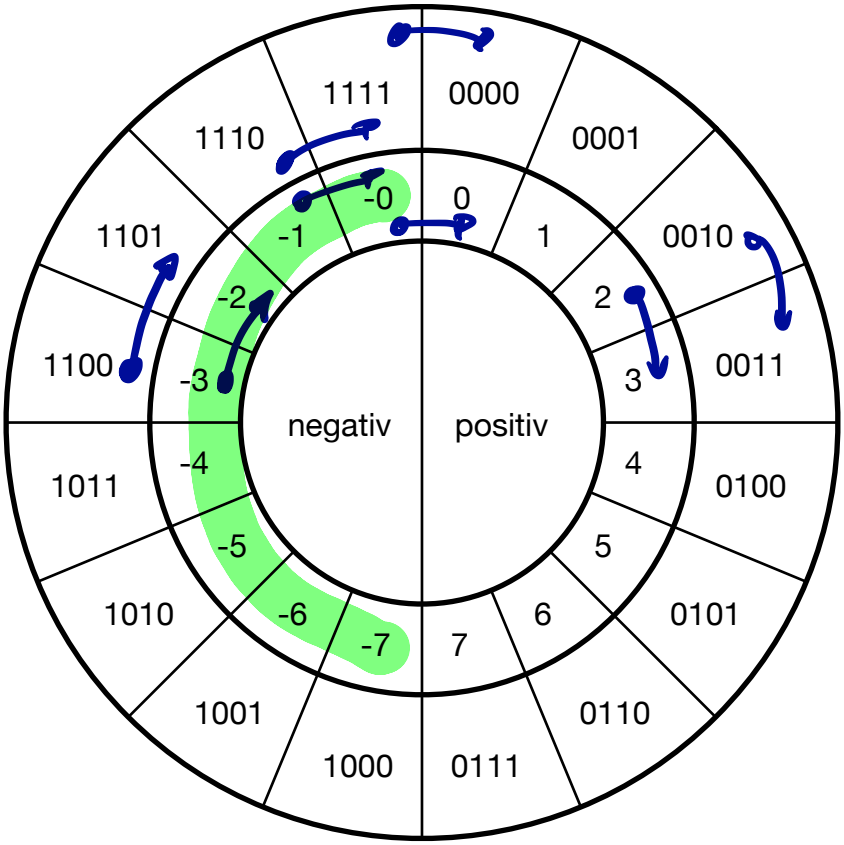
a) Welche Auswirkungen hat es, dass für negative Zahlen die Richtung steigender Werte nicht übereinstimmt?

b) Ist der Wertebereich symmetrisch? Begründung!

Einer-Komplement

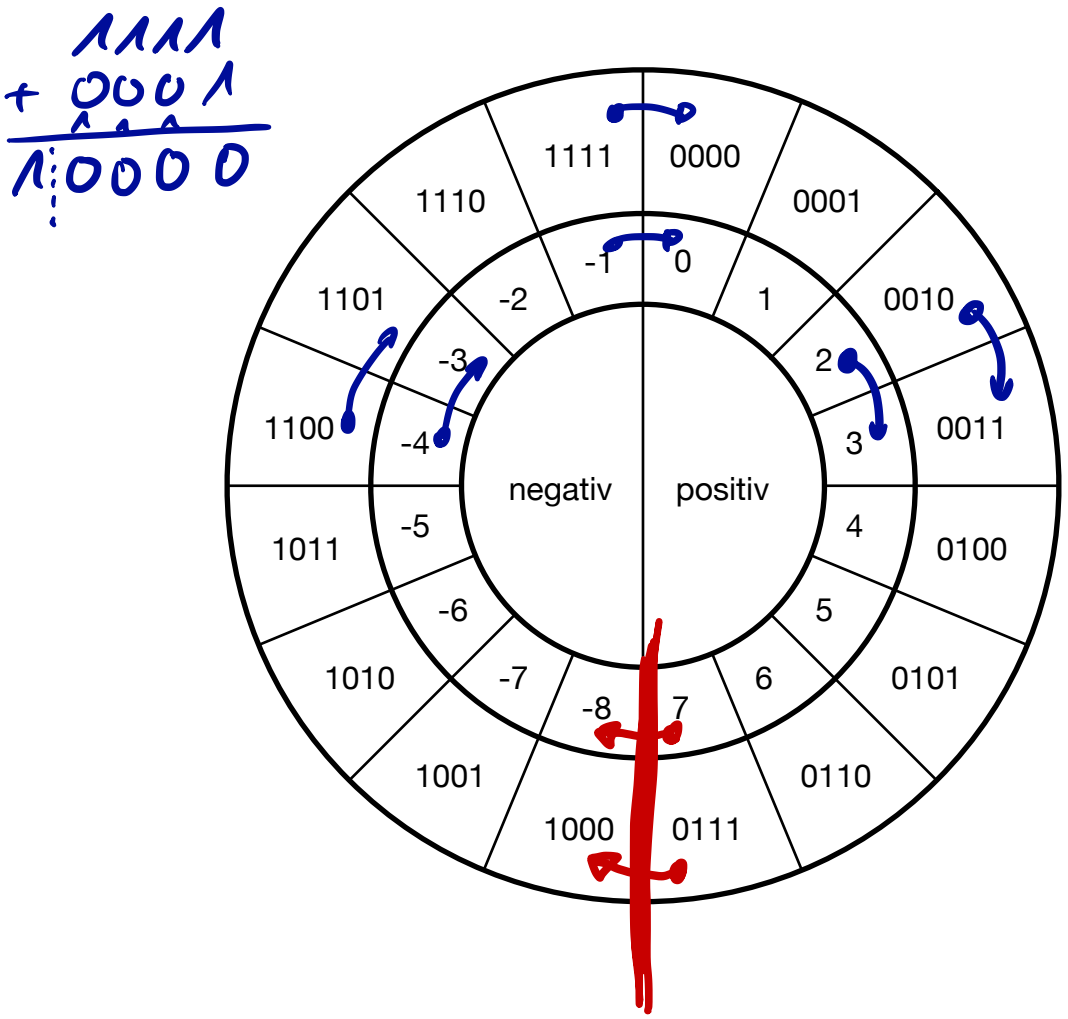
Bei dieser Darstellung werden zur Negierung einer Zahl alle Bits invertiert. Um eine eindeutige Unterscheidung zwischen positiven und negativen Zahlen zu gewährleisten, ist der Betrag der Zahlen auf $2^{n-1} - 1$ beschränkt. Dadurch kann das Vorzeichen der Zahl wieder direkt am MSB abgelesen werden ($0 \Rightarrow$ positiv; $1 \Rightarrow$ negativ).

Der Vorteil dieser Darstellung im Vergleich der Darstellung ‘‘Vorzeichen und Betrag’’ liegt darin, dass die Codierung der negativen Zahlen in derselben Richtung erfolgt wie die Codierung der positiven Zahlen, so dass positive und negative Zahlen auf die gleiche Art und Weise addiert (bzw. subtrahiert) werden können.



Zweier-Komplement

Beim Zweier-Komplement wird zunächst das Einer-Komplement gebildet und dann noch binär der Wert 1 addiert. Auf diese Weise wird die doppelte Null vermieden. Der Wertebereich wird asymmetrisch, was jedoch kein Problem darstellt. Berechnungen können in dieser Codierung mit demselben Algorithmus durchgeführt werden wie im Dezimalsystem. Aus diesem Grund werden vorzeichenbehaftete Festkomma-Zahlen in der Regel im Zweier-Komplement codiert.

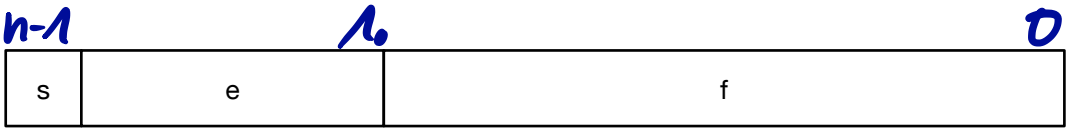


2.5 Codierung von Gleitkommazahlen nach IEEE 754

Durch die fest definierte Kommastelle sind bei Festkommazahlen die Abstände zwischen den einzelnen Zahlenwerten äquidistant. Aus diesem Grund (und aufgrund der endlichen Anzahl an Stellen n) können mit Festkommazahlen nicht gleichzeitig sehr große Zahlen und sehr kleine Zahlen dargestellt werden.
Bei Gleitkommazahlen ist diese Einschränkung aufgehoben. Die Abstände zwischen den einzelnen Zahlenwerten sind um den Wert 0 herum sehr klein. Für große Zahlen werden die Abstände sehr groß, wie in nachstehender Grafik skizziert.

Erreicht wird diese Eigenschaft dadurch, dass die Position des Kommas nicht im Voraus festgelegt ist, sondern in der Zahl durch Angabe eines Exponenten e definiert wird. Der Exponent legt fest, um wieviel die Kommastelle nach links oder rechts verschoben werden muss.

Gleitkommazahlen werden wie folgt codiert:



Bei 32 Bit breiten Gleitkommazahlen (einfache Genauigkeit) gilt die Aufteilung

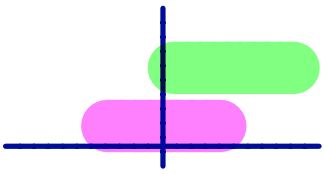
- s = 1 Bit
- e = 8 Bit
- f = 23 Bit,

bei 64 Bit breiten Gleitkommazahlen (doppelte Genauigkeit) gilt die Aufteilung

- s = 1 Bit
- e = 11 Bit
- f = 52 Bit.

Als Wert ergibt sich für

- für *normalisierte* Gleitkommazahlen (Normal-Fall) $v = (-1)^s \cdot 1{,}f \cdot 2^{e-K}$,
- für *de-normalisierte* Gleitkommazahlen (Spezial-Fall) $v = (-1)^s \cdot 0{,}f \cdot 2^{1-K}$.



Die Konstante K hat

- bei einfacher Genauigkeit (32 Bit) den Wert $K = 127$,
- bei doppelter Genauigkeit (64 Bit) den Wert $K = 1023$.

Eine Gleitkommazahl gilt als *normalisiert*, wenn beim Exponenten e weder alle Bits gesetzt noch alle Bits gelöscht sind, d.h.

- $0 < e < 255$ bei 32 Bit
- $0 < e < 2047$ bei 64 Bit.

Eine *denormalisierte* Gleitkommazahl liegt vor, wenn $e = 0$ und gleichzeitig $f > 0$.

Spezialfälle:

- 0:
  - $e = 0$
  - $f = 0$
- $\pm\infty$:
  - s: $+\infty \Rightarrow 0$; $-\infty \Rightarrow 1$
  - e: alle Bits gesetzt $\Rightarrow$ 255 bei 32 Bit, 2047 bei 64 Bit
  - f: alle Bits 0
- NaN (Not a Number)
  - e: alle Bits gesetzt $\Rightarrow$ 255 bei 32 Bit, 2047 bei 64 Bit
  - f: > 0

Aufgaben

Format von Gleitkommazahlen

a) Welchen Wert hat eine Zahl, die in 64 Bit Gleitkomma-Notation mit 0xC028000000000000 codiert wird?