Chair of Embedded Systems
Department of Computer Science
University of Augsburg, Germany

Martin Frieb

Universität Augsburg University

Wintersemester 2020/2021

Informatik 1

Kapitel 3 – Zahlensysteme

Contents

Version 20201110-1431

Als Informatiker sollte man wissen wie Zahlen, Zeichen, Ton, Bilder, usw. durch Bitfolgen **codiert** werden können und auch was man dabei beachten muss (Möglichkeit der **Decodierung**, **Arithmetik** auf codierten Zahlen usw.). Ein Computer kann Daten nur in Form von **1**-en und **0**-en speichern (vgl. Kapitel 2).

Binäres Zahlensystem

Eine Folge von **1**-en und **0**-en ergibt zusammen eine Binärzahl (also eine Zahl im **binären Zahlensystem**). In diesem Kapitel schauen wir uns diese Binärzahlen und ihre mathematischen Grundlagen näher an.

> Definition: 3.13 Binärdarstellung
>
> Die 2-adische Darstellung einer Zahl nennt man **Binärdarstellung**. Eine Zahl in Binärdarstellung besteht aus den Ziffern 0 und 1.

Die B-adische Darstellung ist eine einfache Schreibweise, mit der deutlich wird, in welchem Zahlensystem die jeweilige Zahl notiert ist. Dazu schreibt man die Zahl in Klammern und hinter die Klammer tiefgestellt das jeweilige Zahlensystem, z.B. $(5)_{10}$ entspricht der Zahl Fünf im Dezimalsystem, $(101)_2$ der Zahl Fünf im Binärsystem. Man spricht von einer B-adischen Darstellung, wobei B das Zahlensystem angibt, z.B. 2-adisch für das Binärsystem und 10-adisch für das Dezimalsystem.

Einige Binärzahlen

Für die Dezimalzahlen von 0 bis 15 sollte man die Binärdarstellungen auswendig können:

$0 = (0)_2$	$4 = (100)_2$	$8 = (1000)_2$	$12 = (1100)_2$
$1 = (1)_2$	$5 = (101)_2$	$9 = (1001)_2$	$13 = (1101)_2$
$2 = (10)_2$	$6 = (110)_2$	$10 = (1010)_2$	$14 = (1110)_2$
$3 = (11)_2$	$7 = (111)_2$	$11 = (1011)_2$	$15 = (1111)_2$

Die tiefgestellte Zwei $(X)_2$ nach Angabe der Zahl beschreibt in welchem Zahlensystem die Zahl zu lesen und zu verstehen ist.

Es gilt offenbar $2^n = (10\ldots0)_2$ (mit $n+1$ Stellen) und $2^n-1 = (1\ldots1)_2$ (mit n Stellen)

Wenn Sie das Verfahren in Kapitel 3.2 nachvollzogen haben, führen Sie die Liste oben einfach selbstständig fort.

Weitere Zahlensysteme

Binärzahlen tendieren dazu sehr lang und unübersichtlich zu werden, deswegen bedient man sich aushilfsweise noch weiteren Zahlensystemen: **Oktalsystem** und **Hexadezimalsystem**. Diese schauen wir ins im Lauf des Kapitels auch noch an.

Bedeutung	Operationszeichen	Beispiele
Kleiner	$<$	$1 < 2$
Kleiner gleich	$\leqslant$	$1 \leqslant 1, 1 \leqslant 2$
Größer	$>$	$0 > -1$
Größer gleich	$\geqslant$	$0 \geqslant 0, 0 \geqslant -1$
Gleich	$=$	$0 = 0$
Ungleich	$\neq$	$0 \neq 1$

Bedeutung	Operationszeichen	Beispiele
Addition	$+$	$1 + 2 = 3$
Subtraktion	$-$	$1 - 2 = -1$
Multiplikation	$\cdot$	$1 \cdot 2 = 2$
Reelle Division	$/$	$5/2 = 2.5$
Ganzzahlige Division	$\div$	$5 \div 2 = 2$
Modulo	mod	$5 \bmod 2 = 1$
Abrundung	$\lfloor \cdot \rfloor$	$\lfloor 5.2 \rfloor = 5$
Aufrundung	$\lceil \cdot \rceil$	$\lceil 5.2 \rceil = 6$
Absolutbetrag	$\lvert \cdot \rvert$	$\lvert -6 \rvert = 6$
Positives Vorzeichen	$+$	$+1$
Negatives Vorzeichen	$-$	-1

3.1 Mathematische Grundlagen

Einige mathematische Operationen & Arithmetische Rechenoperationen auf Zahlen

Die Operationszeichen in den beiden Tabellen sind aus der Schule bekannt. Wir werden sie in diesem Kapitel in der Schreibweise, welche in den Tabellen gezeigt wurde, verwenden.

Wichtig: Diese Operationszeichen gelten **nicht** für Programmiersprachen, sondern für mathematische Formeln in Texten. In Programmiersprachen gibt es diese Operationen auch, sie werden aber anders bezeichnet (z.B. wird in C == für den Vergleich auf Gleichheit verwendet)

3.1.1 Mengen

Definition: 3.1 Menge, Element

- Eine **Menge** ist eine Zusammenfassung **unterscheidbarer Objekte** (keine Duplikate) zu einer Gesamtheit.
- Die in einer Menge zusammengefassten Objekte heißen deren **Elemente**. Ist a ein Element einer Menge A, so schreiben wir $a \in A$, andernfalls schreiben wir $a \notin A$.
- Die **leere Menge** hat keine Elemente und wird mit $\emptyset$ bezeichnet.

Beispiel:

Sei $M = \{a, b, c\}$, dann gilt $a \in M$ und $b \in M$ sowie $c \in M$, jedoch nicht $d \in M$. D.h. $d \notin M$. a, b, c sind Elemente der Menge M.

Definition: 3.2 Teilmenge

- Ist jedes Element einer Menge A auch Element einer Menge B, so heißt A **Teilmenge von** B. Ist A eine Teilmenge von B, so schreiben wir $A \subseteq B$.
- Gilt $A \subseteq B$ und $A \neq B$, so heißt A **echte Teilmenge von** B. Ist A eine echte Teilmenge von B, so schreiben wir $A \subset B$.
- Mit $P(A) := \{B \mid B \subseteq A\}$ bezeichnen wir die Menge aller Teilmengen von A, die sog. **Potenzmenge von** A.

Beispiel:

Sei $M = \{a, b, c\}$, dann gilt für:

- **Teilmenge**: $M' = \{a, b\}$ ist eine (echte) Teilmmenge von M, formal geschrieben $M' \subseteq M$.
- **Potenzmenge**:
  Alle Kombinationsmöglichkeiten der Elemente der Menge, inkl. leerer Menge und Menge selbst (da die Potenzmenge *alle* Teilmengen fordert und **keine** *echten* Teilmengen voraussetzt). Somit ist auch stets die ursprüngliche Menge A enthalten. Die Elemente aus M lassen sich wie folgt kombinieren:
  $\{\emptyset\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$
- Somit ist $P(M) = \{\{\emptyset\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$
- Wie man sieht ist die Potenzmenge eine Menge, die Mengen enthält.
- **Beachte**: Reihenfolgen von Elementen spielen in Mengen keine Rolle, d.h $\{b, c\} = \{c, b\}$

Definition: 3.3 Endliche Menge

Eine Menge heißt **endlich**, falls sie endlich viele Elemente hat. Die **Mächtigkeit** $|A|$ einer endlichen Menge A ist die Anzahl ihrer Elemente.

Schreibweise für endliche Mengen:

- Eine endliche Menge A mit $n > 0$ Elementen $a_1, \ldots, a_n$ gibt man durch Aufzählung aller Elemente in geschweiften Klammern in der folgenden Form an: $A = \{a_1, \ldots, a_n\}$.

Beispiel:

- $\{0, 1\}$ ist die Menge mit den Elementen 0 und 1 und hat die Mächtigkeit $|\{0, 1\}| = 2$.
- Die leere Menge hat die Mächtigkeit $|\emptyset| = 0$.

Häufiger ist es aber der Fall, dass wir über unendliche Mengen sprechen, insbesondere bei Definitions- und Wertebereichen bei beispielsweise Funktionen.

Die folgenden festen Symbole für bestimmte (unendliche) Zahlenmengen sollten schon bekannt sein:

- $\mathbb{Z}$: Menge der **ganzen Zahlen**.
- $\mathbb{N} := \{n \mid n \in \mathbb{Z},\ n > 0\}$: Menge der **natürlichen Zahlen**.
- $\mathbb{N}_0 := \{n \mid n \in \mathbb{Z},\ n \geqslant 0\}$: Menge der **nicht-negativen** ganzen Zahlen.
- $\mathbb{Q} := \{\frac{p}{q} \mid p \in \mathbb{Z},\ q \in \mathbb{N}\}$: Menge der **rationalen** Zahlen.
- $\mathbb{R}$: Menge der **reellen** Zahlen.

Definition: Unendliche Menge

- Eine unendliche Menge A gibt man durch **Eigenschaften** $E_1, \ldots, E_n$**, die deren Elemente charakterisieren**, in der folgenden Form an:

$$A := \{a \mid E_1, \ldots, E_n\}.$$

- Dies spricht man wie folgt: *Menge aller Elemente* a ***mit den Eigenschaften*** $E_1, \ldots, E_n$.

- Damit gehören **genau** die Elemente zu A, die **alle** Eigenschaften $E_1, \ldots, E_n$ erfüllen.

Beispiel:

- $A := \{a | a \in \mathbb{N}, a > 5\}$ definiert die Menge A, die alle Elemente aus der Menge $\mathbb{N}$ übernimmt, die gleichzeitig größer als 5 sind. Sie enthält also alle ganzen Zahlen ab der Zahl 6 (also 6,7,8,9, ...).
- Die Menge A hat die Mächtigkeit $|A| = \infty$ (wird im nächsten Abschnitt erklärt), ebenso wie die ursprüngliche Menge $\mathbb{N}$

Mathematische Definitionen

In der obigen Definition und dem zugehörigen Beispiel taucht das bisher noch nicht vorgestellte Symbol := auf, welches im Zusammenhang der Mengenschreibweise benutzt wurde, um einem Bezeichner einen Inhalt zu zuordnen. Formal handelt es sich um das Symbol der **mathematischen Definition**.

Definition: Mathematische Definition

- Eine **mathematische Definition** ist die eindeutige Festlegung der Bedeutung und Verwendung eines Begriffs, Symbols oder einer Schreibweise.
- Ein neues Symbol kann durch Rückführung auf bereits bekannte, vorher definierte Symbole definiert werden.
- Mit dem Zeichen := werden linke und rechte Seite **gleich gesetzt**, wobei
  - Auf der linken Seite von := das neue Symbol steht, das man definieren möchte.
  - Auf der rechten Seite von := die Bedeutung des neuen Symbols festgelegt wird durch eine Formel, in der nur bereits bekannte Symbole vorkommen.
- Zum Beispiel wird in $\mathbb{N} := \{n \mid n \in \mathbb{Z}, n > 0\}$ das neue Symbol $\mathbb{N}$ eingeführt (linke Seite).
- Durch die rechte Seite wird festgelegt, dass es sich bei $\mathbb{N}$ um eine Menge handelt, und zwar um die Menge der **positiven** ganzen Zahlen.
- $\mathbb{N}$ wird damit zurückgeführt auf $\mathbb{Z}$ durch Hinzunahme des zusätzlichen Merkmals **positiv**. Da es auch nicht-positive ganze Zahlen gibt, unterscheiden sich $\mathbb{N}$ und $\mathbb{Z}$.

In dieser Vorlesung benutzen wir := ausschließlich für Definitionen beispielsweise von Intervallen, bei denen wir uns der neuen Mengenschreibweise bedienen.[1]

[1] In einigen Programmiersprachen (z.B. Delphi oder VHDL) wird := auch für Zuweisungen verwendet (in C aber einfach nur =).

Definition: 3.5 Intervalle

Seien $n, m \in \mathbb{R}$ und $n \leqslant m$:

- $[n, m] := \{x \mid x \in \mathbb{R},\ x \geqslant n,\ x \leqslant m\}$.
- $[n, m[:= \{x \mid x \in \mathbb{R},\ x \geqslant n,\ x < m\}$.
- $]n, m] := \{x \mid x \in \mathbb{R},\ x > n,\ x \leqslant m\}$.
- $]n, m[:= \{x \mid x \in \mathbb{R},\ x > n,\ x < m\}$.
- $[n, \infty[:= \{x \mid x \in \mathbb{R},\ x \geqslant n\}$.
- $]n, \infty[:= \{x \mid x \in \mathbb{R},\ x > n\}$.
- $]-\infty, m] := \{x \mid x \in \mathbb{R},\ x \leqslant m\}$.
- $]-\infty, m[:= \{x \mid x \in \mathbb{R},\ x < m\}$.

Man beachte, dass diese Intervalle alle auf $\mathbb{R}$ definiert werden, d.h. es sind auch alle reellen Zahlen zwischen zwei ganzen Zahlen enthalten, z.B. enthält das Intervall $[1, 2]$ auch alle Zahlen, die zwischen 1 und 2 liegen wie 1.5 oder $\sqrt{2}$

Beispiel:

$[1, 4] := \{x | x \in \mathbb{R}, x \geqslant 1, x \leqslant 4\}$
Diese Liste enthält viele Zahlen, z.B. die folgenden: 1, 1.5, 2.2, 3.333, $\frac{5}{3}$, $\sqrt{3}$. Um alle Zahlen anzugeben, wäre man unendlich lange beschäftigt, da alle reellen Zahlen zwischen 1 und 4 enthalten sind.

Da wir nun neben den unendlichen Mengen auch bei Intervallen über Unendlichkeit reden bietet es sich an auch Unendlichkeit formal zu definieren.

Definition: 3.6 Das Symbol ∞

- ∞ bezeichnet einen Wert, die größer ist als jede ganze Zahl (gesprochen **unendlich**).
- $-\infty$ bezeichnet einen Wert, die kleiner ist als jede ganze Zahl (gesprochen **minus unendlich**).

Definition: 3.7 Mengen-Operationen

Seien A, B Mengen.

- Der **Durchschnitt** (die **Schnittmenge**) $A \cap B := \{x \mid x \in A, x \in B\}$ von A und B ist eine Zusammenfassung aller Objekte, die Elemente in A und in B sind.
- Die **Vereinigung(smenge)** $A \cup B := \{x \mid x \in A \text{ oder } x \in B \text{ oder } x \in A \cap B\}$ von A und B ist eine Zusammenfassung aller Objekte, die Elemente in A, in B oder in A und in B sind.
- Die **Differenz** $A \setminus B := \{x \mid x \in A, x \notin B\}$ von A und B ist eine Zusammenfassung aller Objekte, die Elemente in A, aber nicht in B sind.

Beispiel:

- $\{1, 2\} \cap \{2, 3\} = \{2\}$.
- $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$.
- $\{1, 2\} \setminus \{2, 3\} = \{1\}$.

3.2 B-adische Darstellung ganzer Zahlen

Wir betrachten im Rahmen der Vorlesung nicht nur das Binär- und das Dezimalsystem („normales“ Zehnersystem, mit dem man im Alltag rechnet), sondern auch weitere Zahlensysteme. Damit klar ist, in welchem Zahlensystem eine Zahl dargestellt ist, beziehen wir uns immer auf die *Basis* des jeweiligen Zahlensystems, d.h. die Zahl, bei der es im jeweiligen Zahlensystem zum Überlauf kommt und man eine zusätzliche Ziffer braucht (2 für das Binärsystem, 10 für das Dezimalsystem).

Schauen wir uns als Beispiele noch das Oktal- und das Hexadezimalsystem an. Beim Oktalsystem kommt es bei 8 zum Überlauf (d.h. es nutzt die Ziffern 0, 1, 2, 3, 4, 5, 6, 7), beim Hexadezimalsystem bei 16 (hier nutzt man zusätzlich zu den Ziffern 0-9 noch A-F, siehe Details in Definition 3.16).

Definition: 3.14 Oktaldarstellung

Die 8-adische Darstellung einer Zahl nennt man **Oktaldarstellung**. Eine Zahl in Oktaldarstellung besteht aus den Ziffern $0, 1, 2, 3, 4, 5, 6, 7$.

Umrechnung zwischen Binär- und Oktaldarstellung
Wegen $8 = 2^3$ stehen jeweils 3 Binärziffern für eine Oktalziffer:

$(0)_8 = (000)_2$ $(2)_8 = (010)_2$ $(4)_8 = (100)_2$ $(6)_8 = (110)_2$
$(1)_8 = (001)_2$ $(3)_8 = (011)_2$ $(5)_8 = (101)_2$ $(7)_8 = (111)_2$

Beispiel:

Umrechnung zwischen Binär- und Oktaldarstellung
$(174)_8 = (001111100)_2$

Definition: 3.16 Hexadezimaldarstellung

Die 16-adische Darstellung einer Zahl nennt man **Hexadezimaldarstellung**. Eine Zahl in Hexadezimaldarstellung besteht aus den Ziffern $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$ ($A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$).

Schnelle Umrechnung zwischen Binär- und Hexadezimaldarstellung
Wegen $16 = 2^4$ stehen jeweils 4 Binärziffern für eine Hexadezimalziffer:

$(0)_{16} = (0000)_2$ $(4)_{16} = (0100)_2$ $(8)_{16} = (1000)_2$ $(C)_{16} = (1100)_2$
$(1)_{16} = (0001)_2$ $(5)_{16} = (0101)_2$ $(9)_{16} = (1001)_2$ $(D)_{16} = (1101)_2$
$(2)_{16} = (0010)_2$ $(6)_{16} = (0110)_2$ $(A)_{16} = (1010)_2$ $(E)_{16} = (1110)_2$
$(3)_{16} = (0011)_2$ $(7)_{16} = (0111)_2$ $(B)_{16} = (1011)_2$ $(F)_{16} = (1111)_2$

Beispiel:

Umrechnung zwischen Binär- und Hexadezimaldarstellung
$(1B9)_{16} = (000110111001)_2$

3.2.1 Was ist eine B-adische Darstellung?

In der Motivation haben wir bereits informell beschrieben was eine B-adische Darstellung ist, dies definieren wir nun formal:

Definition: B-adische Darstellung ganzer Zahlen

Für jedes $x \in \mathbb{N}$ und für jede ganze Zahl $B > 1$ gibt es eindeutige ganze Zahlen $b_0, \ldots, b_k \in \{0, 1, 2, ..., B-1\}$ mit: $x = \sum_{i=0}^{k} b_i \cdot B^i =: (b_k \ldots b_0)_B$. Wobei $b_0, \ldots, b_k$ einfach die Ziffern der Zahl b von der k-ten bis zur 0-ten Stelle sind.

Man nennt:

- $(b_k \ldots b_0)_B$ die B-**adische Darstellung** von x
- B die **Basis** dieser Darstellung
- $b_0, \ldots, b_k$ die **Ziffern** dieser Darstellung

Die B-adische Darstellung ist also der allgemeine Fall, d.h. die Darstellung einer Zahl in einem beliebigen System, im Gegensatz zu den speziellen Systemen wie beispielswese Binär-, Oktal-, Dezimal- oder Hexadezimalsystem.

Bevor wir uns die Berechnung einer B-adischen Darstellung anschauen, definieren wir erst die Summenschreibweise, von der wir dann Gebrauch machen werden.

Das Summenzeichen Σ zeigt an, dass alle Werte von einem Startwert bis zu einem Endwert zu addieren sind, z.B. $\sum_{i=1}^{4} = 1+2+3+4$. Formal lässt sich das Summenzeichen wie folgt induktiv definieren:

Die Summenschreibweise

Definition:

Wir definieren die Summenschreibweise $\sum_{i=m}^{n} A_i$ mit $m, n \in \mathbb{Z}$, $m < n$, wie folgt **induktiv nach** n:

(1) Induktionsanfang $n = m$: $\sum_{i=m}^{m} A_i := A_m$
(2) Induktionsschritt $n \to n+1$: $\sum_{i=m}^{n+1} A_i := (\sum_{i=m}^{n} A_i) + A_{n+1}$

Beispiel:

Für $m = 1$ und $n = 4$

$$\begin{aligned}
\sum_{i=1}^{4} i &\overset{(2)}{=} (\sum_{i=1}^{3} i) + 4 \\
&\overset{(2)}{=} ((\sum_{i=1}^{2} i) + 3) + 4 \\
&\overset{(2)}{=} (((\sum_{i=1}^{1} i) + 2) + 3) + 4 \\
&\overset{(1)}{=} 1 + 2 + 3 + 4 \\
&= 10
\end{aligned}$$

Definition: Induktive Definition

- Hängt eine Schreibweise von einem Parameter $n \in \mathbb{N}$ ab, so wird diese häufig durch eine sog. **vollständige Induktion nach** n definiert.
- Diese besteht aus 2 Schritten:
  - Definition der Schreibweise für den kleinsten Wert von n (**Induktionsanfang**).
  - Definition der Schreibweise für $n+1$ durch Rückführung auf den Fall n (**Induktionsschritt**).
- Im Beispiel wird die Summenschreibweise zuerst für $n = m$ definiert (Induktionsanfang)
- Dann wird eine Formel angegeben, durch die man die Summenschreibweise für $n + 1$ auf die Summenschreibweise für n (für beliebiges n) zurückführen kann (Induktionsschritt).

3.2.2 Berechnung einer B-adischen Darstellung

Wie kann man die B-adische Darstellung einer Zahl schnell und einfach berechnen?

Im Binärsystem gibt es folgende Stellen: 1er = 2^0, 2er = 2^1 , 4er = 2^2, 8er = 2^3, 16er = 2^4, 32er = 2^5, 64er = 2^6, 128er = 2^7, 256er = 2^8 usw. Die größte Stelle, in die z.B. die Zahl 131 hinein passt, ist 128. Wenn wir von 131 dann 128 abziehen, erhalten wir 3. Die größte Stelle für 3 ist die 2er Stelle. Also ziehen wir von 3 folglich 2 ab, es verbleibt 1. Für 1 bleibt die 1er-Stelle. Als Binärzahl notiert ergibt sich $(10000011)_2$.

Beispiel: B-adische Darstellung

- $21 = 2 \cdot 10^1 + 1 \cdot 10^0 = (21)_{10}$ (Dezimaldarstellung)
- $21 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (10101)_2$ (Binärdarstellung)
- $131 = 2 \cdot 8^2 + 0 \cdot 8^1 + 3 \cdot 8^0 = (203)_8$ (Oktaldarstellung)
- $27 = 1 \cdot 16^1 + 11 \cdot 16^0 = (1B)_{16}$ (Hexadezimaldarstellung)

Die formale Berechnung läuft genau anders herum ab. Sie berechnet erst die letzte Ziffer und am Ende die erste (ein Beispiel dazu folgt gleich weiter unten beim Hornerverfahren):

Beispiel:

Im Binärsystem:	$(\sum_{i=0}^{k} b_i \cdot B^i) \bmod B = b_0$	$131 \bmod 2 = 1$
	$(\sum_{i=0}^{k} b_i \cdot B^i) \div B = \sum_{i=1}^{k} b_i \cdot B^{i-1}$	$131 \div 2 = 65$
	$(\sum_{i=1}^{k} b_i \cdot B^{i-1}) \bmod B = b_1$	$65 \bmod 2 = 1$
	$(\sum_{i=1}^{k} b_i \cdot B^{i-1}) \div B =$	$65 \div 2 = 32$
	$\sum_{i=2}^{k} b_i \cdot B^{i-2}$	$32 \bmod 2 = 0$
	und so weiter	

Führen Sie die Berechnung selbst zu Ende.

Im Oktalsystem:	$(\sum_{i=0}^{k} b_i \cdot B^i) \bmod B = b_0$	$131 \bmod 8 = 3$
	$(\sum_{i=0}^{k} b_i \cdot B^i) \div B = \sum_{i=1}^{k} b_i \cdot B^{i-1}$	$131 \div 8 = 16$
	$(\sum_{i=1}^{k} b_i \cdot B^{i-1}) \bmod B = b_1$	$16 \bmod 8 = 0$
	$(\sum_{i=1}^{k} b_i \cdot B^{i-1}) \div B =$	$16 \div 8 = 2$
	$\sum_{i=2}^{k} b_i \cdot B^{i-2}$	$2 \bmod 8 = 2$
	und so weiter	

Was fällt uns noch auf?
Beobachtung: Wenn wir eine Zahl durch ihre Basis teilen erhalten wir die Zahl um ein Komma verschoben (z.B. im Dezimalsystem $(205)_{10}$ / 10 = $(20.5)_{10}$, im Binärsystem $(100)_2$ / 2 = $(10.0)_2$). Formal notiert: $(\sum_{i=0}^{k} b_i \cdot B^i)/B = b_k \cdot B^{k-1} + \ldots + b_1 \cdot B^0 + b_0 \cdot B^{-1}$

Berechnung der B-adischen Darstellung in C: Hornerverfahren
Nachdem wir formal notiert haben, wie man die B-adische Darstellung einer Zahl findet, können wir diese auch als C-Funktion beschreiben:

```
void horner_nat(int b[], int x, int B)
{
    int i = 0;
    while (x > 0) {
        b[i] = x % B;
        x = x / B; /*ganzzahlige Division!*/
    ++i;
    }
}
```

In Worten lässt sich das Hornerverfahren so zusammenfassen:

- Es werden die zu konvertierende Zahl x, die Basis b und ein freies Ergebnisfeld b übergeben.
- Wir definieren uns eine Zählervariable i, welche gleichzeitig auch dazu dient, im Ergebnisfeld b an die nächste Stelle zu schreiben.
- Nun wiederholen wir die folgenden Schritte solange unsere aktuell zu konvertierende Zahl größer als 0 ist, d.h. noch Stellen übrig sind:
  - An der aktuellen Stelle im Ergebnisfeld wird das Ergebnis der Modulo-Funktion (Division mit Rest) von x mit der Basis B gespeichert.
  - Anschließend wird x aktualisiert. Dazu wird eine ganzzahlige Division von x durch die Basis B durchgeführt.
- Das Verfahren terminiert[2], wenn wir durch eine Division von x mit der Basis B ein neues x erhalten welches sich dann zwischen 0 und 1 befindet, bzw. wenn x kleiner als die Basis B ist.

Beispiel:

$x = 131, B = 8$

$i = 0$: $b_0 = 131 \bmod 8 = 3$, $x = 131 \div 8 = 16$
$i = 1$: $b_1 = 16 \bmod 8 = 0$, $x = 16 \div 8 = 2$
$i = 2$: $b_2 = 2 \bmod 8 = 2$, $x = 2 \div 8 = 0$

Rechnen in einer B-adischen Darstellung
B-adische Darstellungen von Zahlen mit $B \neq 10$ können auf dieselbe Art addiert, subtrahiert, multipliziert und dividiert werden wie im Dezimalsystem durch **stellenweise Ausführung der Operation mit Übertrag**. Beachte dabei: Der Übertrag erfolgt hier bei Ergebnissen $> B-1$ (denn dann entsteht ein Beitrag zur Ziffer der nächsthöheren B-Potenz)

Beispiele: Stellenweise Addition und Subtraktion mit Übertrag
Im Grunde ähnelt die stellenweise Addition und Subtraktion in den neuen Zahlensystemen (Binär-, Okta- und Hexadezimalsystem) dem stellenweisen Rechnen im Dezimalsystem, welches aus früheren Schuljahren bekannt sein soll.

Zur Wiederholung:

Die Zahlen werden untereinander geschrieben, so dass die gleichen Stellen übereinander stehen. Nun werden von hinten nach vorne entlang jeder Spalte die beiden Zahlen addiert bzw. subtrahiert und das Ergebnis darunter vermerkt. Hier gilt es nun darauf zu achten, dass sich das Ergebnis in den Grenzen der erlaubten Zahlen pro Stelle befindet. Im Dezimalsystem beispielsweise zwischen den Zahlen von von 0 bis 9, da die Basis 10 ist. Zusätzlich wird in einer weiteren Zeile vermerkt ob es bei der Rechnung zu einem Übertrag kommt, also ob die Addition bzw. Subtraktion dazu geführt hat, dass der Zahlenbereich einmal verlassen wurde. Pro Bereichsüberlauf wird also im Übertrag

[2] Ein Programm *terminiert*, wenn wir das Ende erreichen, z.B. in einer `main`-Funktion das abschließende `return 0;` ausgeführt wird.

eine 1 bei nächst-höheren Stelle vermerkt. Dieses Verfahren wird für jede Stelle wiederholt, wobei auch der Übertrag der vorhergehenden Stelle berücksichtigt werden muss.

Beispiel:

$(451)_8 + (337)_8$:

Stelle			2	1	0
Ziffer			8^2	8^1	8^0
a		0	4	5	1
b	+	0	3	3	7
Übertrag			1	1	1
a + b		1	0	1	0

Beispiel:

$(451)_8 - (337)_8$:

Stelle		2	1	0
Ziffer		8^2	8^1	8^0
a		4	5	1
b	−	3	3	7
Übertrag		0	1	0
a − b		1	1	2

Beispiel:

$(111)_2 + (11)_2$:

Stelle		3	2	1	0
Ziffer		2^3	2^2	2^1	2^0
a			1	1	1
b	+			1	1
Übertrag		1	1	1	0
a + b		1	0	1	0

3.3 Oktal- und Hexadezimaldarstellungen in C

3.3.1 Weitere Schreibweisen für Ganzzahl-Konstanten

In der Programmierung sind Ganzzahl- und Zeichenkonstanten sehr wichtig, um für eine bessere Lesbarkeit zu sorgen und mehr Ordnung und Struktur in ein Programm zu bringen.

Hierbei gibt es verschiedenste Möglichkeiten, Zahlen und Buchstaben darzustellen.

Zeichenkonstanten
In der C-Programmierung gibt es für jeden Buchstaben (Character, kurz `char`) auch eine Zahl, womit man dieses Zeichen darstellen kann. Das bedeutet, dass man bspw. für den Buchstaben 'A' auch die Zahl 65 nutzen kann. Welcher Buchstabe nun zu welcher Zahl gehört, lässt sich in einer *ASCII Tabelle*[3] nachschlagen (und müssen nicht auswendig gelernt werden ;))

Natürlich funktioniert das auch umgekehrt – die Zahl 65 kann auch dem Buchstaben 'A' entsprechen.

Man sollte aber beachten, dass man Zahlen nicht nur für Buchstaben verwenden will, sondern auch als das, was sie sind: Zahlen. Damit der Computer / der Compiler (und wir) nun wissen, wie man eine Zahl interpretieren soll, muss man den jeweiligen Datentypen spezifizieren.

Beispiel:

```c
int main(void)
{
    /* Speichert die Zahl 65 */
    int zahl = 65;
    int zahl2 = 'A';

    /* Speichert den Buchstaben 'A' */
    char character = 65;
    char character2 = 'A';

    /* Gibt den Buchstaben 'A' aus */
    printf("%c", zahl);

    /* Gibt den Buchstaben 'A' aus */
    printf("%c", character);

    /* Gibt die Zahl 65 aus */
    printf("%i", zahl);

    /* Gibt die Zahl 65 aus */
    printf("%i", character);
```

[3] siehe z.B. `https://www.torsten-horn.de/techdocs/ascii.htm`

```
    return 0;
}
```

Man ist hierbei natürlich nicht auf einfache `print`-Befehle o.ä. beschränkt. Man kann damit auch Vergleiche ziehen, Schleifen benutzen, etc.

Beispiel:

```
int main(void)
{
    /* Initialisiere und generiere eine
       Zufallszahl */
    int x;

    srand(time(NULL));
    x = rand();

    /*
    Ist diese Zufallszahl zwischen 'A' oder 'Z',
        gebe ich sie aus.
    Dafür hat man zwei Möglichkeiten.

    Fall 1: die unleserliche Variante:
    */
    if (x >= 65 && x <= 90) {
        printf("%c", x);
    }
    /*
    Man hat irgendwelche Zahlen im Raum und weiß
        nicht, für was sie stehen sollen.

    Deswegen:
    Fall 2: die leserliche Variante
    */
    if (x >= 'A' && x <= 'Z') {
        printf("%c", x);
    }

    return 0;
}
```

Zahlenkonstanten

Bei solchen Umwandlungen ist man natürlich auch nicht auf Zeichen beschränkt, sondern kann das selbe Spiel auch mit Zahlen treiben.

Es gibt auch Situationen, wo man direkt mit Oktal- oder Hexadezimalzahlen arbeiten will. Zwar ist die Verwendung von Oktalzahlen heutzutage nicht mehr so üblich, jedoch werden Hexadezimalzahlen immernoch in der hardwarenahe Programmierung verwendet.

Die Programmiersprache C bietet hierbei nützliche Schreibweisen an, wie man Oktal- und Hexadezimalzahlen darstellen kann.

Definition: 3.18 Oktal-Schreibweise von Ganzzahl-Konstanten

`<Vorzeichen>0<Ziffernfolge>`
wobei:

- Das Vorzeichen ist + oder – und ist optional.
- Die Ziffernfolge enthält nur Oktalziffern und entspricht der Oktaldarstellung der Zahl
- Die Ziffernfolge enthält keine führenden `0`en

Die Angabe einer Zahl in der Oktaldarstellung ist nicht schwer. Man muss nur das Vorzeichen richtig setzen, eine 0 (die Zahl Null) schreiben und danach die Oktalzahl selber angeben.

Beispiel:

- `02` entspricht 2 im Dezimalsystem
- `033` entspricht 27 im Dezimalsystem

Zugehörige Umwandlungsangabe für `printf`: `%o`

Beispiel:

```
int main(void)
{
    int oktalzahl = 033;

    /* Gibt die Dezimalzahl 27 aus;
       da mit %i eine Dezimalzahl
       gefordert wird */
    printf("%i", oktalzahl);

    /* Gibt die Oktalzahl 33 aus; da
        mit %o eine Oktalzahl
       gefordert wird */
    printf("%o", oktalzahl);
    printf("%o", 27);

    return 0;
}
```

Dasselbe Vorgehen kann man natürlich auch für Hexadezimalzahlen verwenden. Die einzige Änderung liegt hier am Präfix. Anstatt einer führenden Null verwendet man ein führendes `0x`

Definition: Hexadezimal-Schreibweise von Ganzzahl-Konstanten

`<Vorzeichen>0x<Ziffernfolge>`
wobei:

- Das Vorzeichen ist + oder − und ist optional.
- Die Ziffernfolge enthält nur Hexadezimalziffern und entspricht der Hexadezimaldarstellung der Zahl
- Die Ziffernfolge enthält keine führenden `0`en
- `x` kann auch groß geschrieben werden:
  `<Vorzeichen>0X<Ziffernfolge>`

Beispiel:

Beispiele:

- `0x2` entspricht 2 im Dezimalsystem
- `0x1B` entspricht 27 im Dezimalsystem

Zugehörige Umwandlungsangaben für `printf`: `%x` oder `%X`

Beispiel:

```c
int main(void)
{
    int hexadezimalzahl = 0x1B;

    /* Gibt die Dezimalzahl 27 aus;
       da mit %i eine Dezimalzahl
       gefordert wird */
    printf("%i", dezimalzahl);

    /* Gibt die Hexadezimalzahl 1B
       aus; da mit %x eine
       Hexadezimalzahl gefordert
       wird */
    printf("%x", hexadezimalzahl);
    printf("%x", 27);

    return 0;
}
```

Wie schon bei den Zeichenkonstanten auf Seite 15 angesprochen, können Buchstaben als Zahlen

dargestellt und Zahlen als Buchstaben dargestellt werden. Das funktioniert für Dezimalzahlen, aber natürlich auch für Oktal- und Hexadezimalzahlen.
Um einen `char` mit einer Oktal- bzw. Hexadezimalzahl darzustellen, muss man diesen wie folgt aufschreiben.

Definition: 3.20 Oktal-Schreibweise von ASCII-Konstanten

`'\<Ziffernfolge>'`
wobei:

- Die Ziffernfolge enthält nur Oktalziffern und entspricht der **Oktaldarstellung des ASCII-Codes** des Zeichens

Beispiel:

`'\101'` entspricht `'A'`, `'\0'` ist die binäre Null

Beispiel:

```
int main(void)
{
    char oktalwert = '\101';

    /* Gibt die Dezimalzahl 65 aus;
       da mit %i eine Dezimalzahl
       gefordert wird */
    printf("%i", oktalwert);

    /* Gibt das Zeichen 'A' aus; da
       mit %c ein char gefordert
       wird */
    printf("%c", oktalwert);

    return 0;
}
```

Und natürlich gilt das selbe Spiel auch für Hexadezimalzahlen.

Definition: 3.21 Hexadezimal-Schreibweise von ASCII-Konstanten

`'\x<Ziffernfolge>'`
wobei:

- Die Ziffernfolge enthält nur Hexadezimalziffern und entspricht der **Hexadezimaldarstellung des ASCII-Codes** des Zeichens

Beispiel:

'\x41' entspricht 'A'

Beispiel:

```
int main(void)
{
    int hexwert = '\x41';

    /* Gibt die Dezimalzahl 65 aus;
       da mit %i eine Dezimalzahl
       gefordert wird */
    printf("%i", hexwert);

    /* Gibt das Zeichen 'A' aus; da
       mit %c ein char gefordert
       wird */
    printf("%c", hexwert);

    return 0;
}
```

3.4 B-adische Darstellung reeller Zahlen

Dem pfiffigen Leser bzw. der pfiffigen Leserin ist sicherlich schon aufgefallen, dass wir uns im bisherigen Verlauf nur auf ganze Zahlen beschränkt haben. Es war also nie die Rede von rationalen (oder sogar reellen) Zahlen wie bspw. $(12.5)_8$. Das wird jedoch spätestens dann zum Problem, wenn man an der Kasse steht und die zu zahlende 1.99€ in eine coole B-adische Form bringen will.

Was ist die B-adische Darstellung reeller Zahlen?
Jeder kann eine reelle Dezimalzahl darstellen. Dazu muss man nur ein Komma innerhalb der Zahlen einfügen.

Beispiel:

Reelle Dezimalzahl: 1.3
;)

Für B-adische Zahlen funktioniert das analog.

Will man das nun formaler definieren, kommt man zur Definition 3.22:

Definition: 3.22 B-adische Darstellung reeller Zahlen

Für jede reelle Zahl $r \in]0,1[$ und für jede ganze Zahl $B > 1$ gibt es eindeutige ganze Zahlen $b_i \in \{0,1,2,...,B-1\}$, $i \in \{n \mid n \in \mathbb{Z}, n \leqslant -1\}$, mit:

$$r = \sum_{i=-1}^{-\infty} b_i \cdot B^i =: (0\,.\,b_{-1}b_{-2}\ldots)_B.$$

Man nennt:

- $(0\,.\,b_{-1}\ldots)_B$ die B-**adische Festkomma-Darstellung** von r mit **Nachkommastellen** $b_{-1}, b_{-2}, \ldots$ (die Anzahl der Nachkommastellen kann unendlich sein)
- B die **Basis** dieser Darstellung
- $\ldots, b_{-2}, b_{-1}$ die **Ziffern** dieser Darstellung

(auf eine formale Definition unendlicher Summen wird hier verzichtet - siehe Mathematik für Informatiker)

Berechnung der Basisdarstellung reeller Zahlen

Nun wollen wir auch reelle Zahlen vom Dezimalsystem in eine beliebige B-adische Darstellung umkonvertieren.

Die Werte links vom Komma werden wie gerade im Kapitel 3.2 gelernt in die B-adische Darstellung gebracht. Daran hat sich nichts verändert.
Die Nachkommastellen werden gesondert behandelt. Hierzu eignet es sich, die Konvertierung anhand eines Beispieles zu erklären:

Beispiel:

Das Ziel ist nun, exemplarisch anhand der Konvertierung von $(18.69189453125)_{10}$ in die 16-adische Darstellung (Hexadezimaldarstellung) die Berechnung der Basisdarstellung reeller Zahlen zu erklären. (Das Beispiel mag zwar eine sehr komische Zahl sein, aber die daraus resultierenden Werte sind schöner :))

Die Werte links vom Komma, also die 18, werden wie gewohnt in das Hexadezimalsystem konvertiert.

$$(18)_{10} = (12)_{16}$$

Die Werte rechts vom Komma sind nun interessanter. Diese Nachkommastellen werden zuerst mit dem Wert B multipliziert. In unserem Fall ist $B = 16$:

$$0.69189453125 \cdot 16 = 11.0703125$$

Man bekommt wieder eine Zahl (die 11) gefolgt von Nachkommastellen (die 0.0703125). Die Zahl vor dem Komma lässt sich wie gewohnt in die B-adische

Darstellung konvertieren:

$$(11)_{10} = (\mathrm{B})_{16}$$

Dieses $(\mathrm{B})_{16}$ ist nun unsere erste Nachkommastelle. Allerdings sind wir hier noch nicht fertig, da immer noch der Rest aus der obigen Rechnung bestehen bleibt. Deswegen wiederholen wir den Schritt, und multiplizieren jetzt 0.0703125 mit 16:

$$0.0703125 \cdot 16 = 1.125$$

Die Zahl links vom Komma bildet nun die 2. Nachkommastelle unserer Zahl von oben ($(1)_{10} = (1)_{16}$).

Da immer noch ein Rest besteht, machen wir weiter:

$$0.125 \cdot 16 = 2.0$$

Die 2 bildet unsere 3. Nachkommastelle ($(2)_{10} = (2)_{16}$) und wir sehen, dass keine Zahl mehr nach dem Komma kommt. Das heißt, wir sind fertig und können endlich unsere Hexadezimalzahl angeben:
$(18.69189453125)_{10} = (12.\mathrm{B}12)_{16}$

Beispiel:

Hier nochmal einige kürzere Beispiele:

- $27.75 = 2 \cdot 10^1 + 7 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2} = (27.75)_{10}$
- $27.75 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = (11011.11)_2$
- $27.75 = 3 \cdot 8^1 + 3 \cdot 8^0 + 6 \cdot 8^{-1} = (33.6)_8$
- $27.75 = 1 \cdot 16^1 + 11 \cdot 16^0 + 12 \cdot 16^{-1} = (1\mathrm{B}.\mathrm{C})_{16}$

Formal aufgeschrieben funktioniert die Berechnung der Nachkommastellen wie folgt:

Definition:

Sei $r := (0.b_{-1}b_{-2}\ldots)_B$:
$(0.b_{-1}b_{-2}\ldots)_B \cdot B = (\sum_{i=-1}^{-\infty} b_i \cdot B^i) \cdot B = \sum_{i=0}^{-\infty} b_{i-1} \cdot B^i =$
$(b_{-1}.b_{-2}\ldots)_B$:

- $(b_{-1})_b$ = Vorkommastellen von $(r \cdot b)_{10}$
  $b = 2$: $(1)_2$ = Vorkommastellen von $(0.75 \cdot 2)_{10} = (1.5)_{10}$
  $b = 16$: $(c)_{16}$ = Vorkommastellen von $(0.75 \cdot 16)_{10} = (12.0)_{10}$
- $(b_{-2})_b$ = Vorkommastellen von $((r \cdot b - (b_{-1})_b) \cdot b)_{10}$
  $b = 2$: $(1)_2$ = Vorkommastellen von $((0.75 \cdot 2 - 1) \cdot 2)_{10} = (1.0)_{10}$
  $b = 16$: $(0)_{16}$ = Vorkommastellen von $((0.75 \cdot 16 - 12) \cdot 2)_{10} = (0.0)_{10}$
- und so weiter, solange es weitere Nachkommastellen gibt
  (fertig, da keine weiteren Nachkommastellen)

Berechnung der Basisdarstellung reeller Zahlen in C: Erweitertes Hornerverfahren

Nachdem wir Vorgehensweise formal nachvollzogen haben, schreiben wir das nun in Code auf; nämlich als *Erweitertes Hornerverfahren* (muss **nicht** terminieren[4]):

```
1  void horner_dec(int b[], double r, int B)
2  {
3      int i = 1;
4      while (r > 0) {
5          b[i] = floor(r * B); /*Vorkommastellen*/
6          r = r * B - b[i]; /*Nachkommastellen*/
7          ++i;
8      }
9  }
```

Das erweiterte Hornerverfahren arbeitet hier ähnlich zum Hornerverfahren aus Kapitel 3.2. Der einzige Unterschied ist, dass das Erweiterte Hornerverfahren für die Nachkommastellen funktioniert, also Zahlen rechts vom Komma. Für die Zahlen links vom Komma wendet man das bereits bekannte Hornerverfahren an.
Der große (und entscheidende) Unterschied befindet sich innerhalb der `while`-Schleife. Dort lässt sich der Algorithmus folgendermaßen zusammenfassen:

- Zeile 1: Die Zahl `r` soll in eine `B`-adische Darstellung gebracht werden. Das Ergebnis wird in `b[]` gespeichert.
  Beachte: $r < 1$. Also soll die Zahl `r` zwischen 0 und 1 liegen.
- Zeile 5: Damit berechnet man die aktuelle Zahl der Nachkommastelle. Man rechnet `r` $\cdot$ `B` und rundet dann das Ergebnis ab (Bspw. $0.4 * 3$ wird zu 1)
- Zeile 6: In dieser Zeile wird `r` für die nächste Iteration angepasst. Man rechnet wieder `r` $\cdot$ `B`, aber zieht von diesem das gerade berechnete Ergebnis ab. Das bedeutet, die Zahl links vom Komma wird „weggeschmissen“ und durch eine 0 ersetzt (bspw. $0.4 * 3$ wird zu 0.2)

[4] Die `while`-Schleife läuft evtl. unendlich lange, da es unendlich viele Nachkommastellen geben kann.

- Zeile 7: Nun erhöht man die Zählvariable `i` und macht solange weiter, bis `r` gleich 0 ist. (Hier gibt es aber keine Garantie für Terminierung. Es kann sein, dass der Algorithmus "unendlich lange„ läuft).

Beispiel: $r = 0.75$, $B = 2$

$i = 1$: $(b_{-1})_2 = \lfloor 0.75 \cdot 2 \rfloor = 1$, $r = 0.75 \cdot 2 - 1 = 0.5$
$i = 2$: $(b_{-2})_2 = \lfloor 0.5 \cdot 2 \rfloor = 1$, $r = 0.5 \cdot 2 - 1 = 0$

Bei der Berechnung der Nachkommastellen zu einer Basis B kann es zu einer **periodischen Wiederholung** kommen:

Beispiel: $r = 0.5$, $B = 3$

$(b_{-1})_3 = \lfloor 0.5 \cdot 3 \rfloor = 1$, $r = 0.5 \cdot 3 - 1 = 0.5$
$(b_{-2})_3 = \lfloor 0.5 \cdot 3 \rfloor = 1$: Periodische Wiederholung

Ergebnis:

$$0.5 = (0.\overline{1})_3$$

3.4.1 Rechnen mit der Basisdarstellung reeller Zahlen

Addition / Subtraktion:
Addition und Subtraktion zweier reellen Zahlen in der Basisdarstellung funktionieren analog wie mit ganzen Zahlen. Der einzige Unterschied ist, dass man nun Kommazahlen vor sich hat.

Beispiel:

Stellenweise mit Übertrag wie bei ganzen Zahlen. Beispiel für die Berechnung von $a - b$, mit $B = 8$:

Stelle		3	2	1	0	.	−1	−2
Ziffer		8^3	8^2	8^1	8^0	.	8^{-1}	8^{-2}
a			7	2	4	.	7	6
b	−			3	6	.	2	2
Übertrag		0	1	1	0	.	0	0
$a - b$		0	6	6	6	.	5	4

Multiplikation / Division
Will man einen Wert in der B-adischen Darstellung mit dem Wert B multiplizieren bzw. dividieren, funktioniert das analog wie im bekannten 10er System.
Rechnet man bspw. $1.2 \cdot 10$ im Dezimalsystem, verschiebt man nur das Komma um eine Stelle nach rechts. Genauso funktioniert das auch in der B-adischen Darstellung.

Will man das nun etwas formeller beschreiben, bekommt man folgendes:

- Multiplikation:

$$\begin{aligned}
&(b_k \dots b_0 \,.\, b_{-1} b_{-2} \dots)_B \cdot B \\
&= (\sum_{i=k}^{-\infty} b_i \cdot B^i) \cdot B \\
&= \sum_{i=k}^{-\infty} b_i \cdot B^{i+1} \\
&= b_k \dots b_0 b_{-1} \,.\, b_{-2} \dots)_B
\end{aligned}$$

- Division:

$$\begin{aligned}
&(b_k \dots b_1 b_0 \,.\, b_{-1} \dots)_B / B \\
&= (\sum_{i=k}^{-\infty} b_i \cdot B^i) / B \\
&= \sum_{i=k}^{-\infty} b_i \cdot B^{i-1} \\
&= (b_k \dots b_1 \,.\, b_0 b_{-1} \dots)_B
\end{aligned}$$

Das sieht bei **ganzen Zahlen** so aus:

- $(b_k \dots b_0)_B \cdot B = (b_k \dots b_0 0)_B$
- $(b_k \dots b_1 b_0)_B / B = (b_k \dots b_1 \,.\, b_0)_B$

> Beispiel:
>
> - $(101.01)_2 \cdot 2 = (1010.1)_2$
> - $(500.1)_{10} \cdot 10 = (5001.0)_{10}$
> - $(5A.1E)_{16} / 16 = (5.A1E)_{16}$
> - $(500.1)_{10} / 10 = (50.01)_{10}$

3.4.2 Gleitkommadarstellung

Wie vieles im Leben sind nun aber auch reelle Zahlen im Computer normiert, damit man nicht hunderte verschiedene Schreib- und Darstellungsweisen besitzt. Diese Normierung wird *Gleitkommadarstellung* genannt (mehr dazu in der Norm IEEE 754):

Definition: 3.26 Gleitkommadarstellung

Für eine reelle Zahl $r \in \mathbb{R}$ und eine ganze Zahl $B > 1$ ist

$$r = m \cdot B^e$$

eine B-**adische Gleitkomma-Darstellung** von r mit **Mantisse** m und **Exponent** e, wobei m eine B-adische Festkomma-Zahl und e eine ganze Zahl ist.

Einfach gesagt bedeutet das nun, dass man die Zahl r einfach nur in eine andere Form bringt. Nämlich wird r nun als Produkt abgebildet, von einer Zahl m (m wird *Mantisse* genannt) multipliziert mit B (B gibt an, in welcher B-adischen Darstellung wir uns befinden). B besitzt dabei auch einen Exponenten e, welcher auch nur eine ganz normale Zahl ist.

Beispiel:

Wir wollen nun beispielsweise die Zahl $(31.4)_{10}$ in die Gleitkommadarstellung umformen:
Das B ist in unserem Fall 10, da wir uns innerhalb der Dezimalzahlen befinden.
An dieser Stelle ist noch keine feste Form für die Gleitkommazahl festgelegt (dies machen wir gleich im Abschnitt 3.4.2 "Normierte Gleitkommazahlen"). Deswegen wählen wir einfach mal $e = 2$.

Nun müssen wir überlegen, wie das m zu wählen ist, sodass gilt:

$$31.4 = m \cdot 10^2$$

Das sollte kein großes Problem darstellen und man bekommt für den Wert m: $m = 0.314$. Somit haben wir alle Werte und eine Gleitkommadarstellung unserer Zahl:

$$31.4 = 0.314 \cdot 10^2$$

Jedoch wird im oberen Beispiel ein Problem angesprochen, das für Gleitkommadarstellungen existiert: Das e ist frei wählbar. Das bedeutet, es gibt unendlich viele Gleitkommadarstellung für eine Zahl.

Beispiel:

Reelle Zahlen haben unendlich viele verschiedene Gleitkommadarstellungen in jeder Basis:

- $12.53 = 12.53 \cdot 10^0 = 1.253 \cdot 10^1 = 0.1253 \cdot 10^2 = 125.3 \cdot 10^{-1}$
- $3.25 = (11.01)_2 \cdot 2^0 = (1.101)_2 \cdot 2^1 = (110.1)_2 \cdot 2^{-1}$

Normierte Gleitkommadarstellung

Um dem Problem zu begegnen, hat man die *normierte Gleitkommadarstellung* eingeführt. Die Idee dahinter ist die selbe wie in der Definition 3.26, bloß haben wir eine Beschränkung: $\mathfrak{m}$ muss größer gleich 1 und kleiner als B sein.

> Definition: Normierte Gleitkommadarstellung
>
> Eine B-adische Gleitkomma-Darstellung
>
> $$r = \mathfrak{m} \cdot B^e$$
>
> heißt **normierte B-adische Gleitkomma-Darstellung** von r,
> falls $1 \leqslant |\mathfrak{m}| < B$.

$|\mathfrak{m}|$ meint hierbei den Betrag von $\mathfrak{m}$, d.h. $\mathfrak{m}$ könnte auch negativ sein.

> Beispiel:
>
> Um nun nochmal das Beispiel von oben aufzugreifen:
> Wir wollen nun beispielsweise die Zahl $(31.4)_{10}$ in die *normierte* Gleitkommadarstellung umformen:
> Das B ist in unserem Fall 10, da wir uns innerhalb der Dezimalzahlen befinden.
> Es gilt nun die Beschränkung, dass $1 \leqslant |\mathfrak{m}| < B$. Wir sind nun sehr stark in der Wahl vom e beschränkt, da durch diese Einschränkung gelten muss:
>
> $$31.4 = 3.14 \cdot 10^e$$
>
> Man wählt nun $e = 1$ um die Gleichung zu erfüllen, und bekommt somit eine normierte Gleitkommadarstellung von
>
> $$31.4 = 3.14 \cdot 10^1$$

Um nun noch einige weitere Beispiele anzuführen:

> Beispiel:
>
> Jede reelle Zahl hat **genau eine** normierte Gleitkommadarstellung in jeder Basis:
>
> - $12.53 = 1.253 \cdot 10^1$
> - $3.25 = (1.101)_2 \cdot 2^1$

Rechnen mit normierten Gleitkommadarstellungen

Wir nehmen zur Vereinfachung an:

- $r_1 = \mathfrak{m}_1 \cdot B^{e_1}$ normiert mit $\mathfrak{m}_1 > 0$.
- $r_2 = \mathfrak{m}_2 \cdot B^{e_2}$ normiert mit $\mathfrak{m}_2 > 0$.

D.h.: r_1 und r_2 sind „ganz normale“, beliebige positive Zahlen in der Gleitkommacodierung

Will man nun zwei Zahlen **vergleichen**:

1. Es gilt $r_1 < r_2$ genau dann wenn (gdw.):
   - Entweder der Exponent von r_1 kleiner ist als der von r_2: $e_1 < e_2$
   - Oder wenn der Exponent der beiden Zahlen übereinstimmt, dann muss die Mantisse von r_1 kleiner sein als die von r_2: $e_1 = e_2 \wedge m_1 < m_2$
2. Die beiden Zahlen sind identisch, wenn sowohl der Exponent als auch die Mantissen übereinstimmen: $e_1 = e_2 \wedge m_1 = m_2$

> Beispiel:
>
> - $9.999 \cdot 10^1 < 1 \cdot 10^2$ (Der Exponent der linken Zahl (Exponent = 1) ist kleiner als der Exponent der rechten Zahl (Exponent = 2))
> - $(1.1)_2 \cdot 2^1 < (1.11)_2 \cdot 2^1$ (Die Exponenten beider Zahlen sind gleich (Exponent = 1). Somit vergleicht man deren Mantisse)

Addition/Subtraktion
Die Addition/Subtraktion von Zahlen in normierter Gleitkomma-Darstellung läuft in folgenden Schritten ab:

1. Exponentenangleich: Bringe beide Zahlen auf den selben Expontenten. Verschiebe dazu das Komma einer der beiden Zahlen und passe ihren Exponenten entsprechend an.
2. Addition/Subtraktion der beiden Mantissen: Da der Exponent bei beiden Zahlen nun identisch ist, kann man die Mantissen einfach addieren bzw. subtrahieren.
3. Normierung (falls nötig): Bringe das Ergebnis wieder in die normierte Gleitkomma-Darstellung mit einem Exponenten $1 \leqslant |m| < B$, falls es nicht schon so da steht.

Addition für $r_1 \geqslant r_2$ (alle Rechenschritte in der Basis B):

$$
\begin{aligned}
& r_1 + r_2 \\
&= m_1 \cdot B^{e_1} + (m_2 \cdot B^{e_2 - e_1}) \cdot B^{e_1} && \text{(Exponentenangleich)} \\
&= (m_1 + m_2 \cdot B^{e_2 - e_1}) \cdot B^{e_1} && \text{(Addition der Mantissen)} \\
&= m \cdot B^{e} && \text{(Normierung - falls nötig)}
\end{aligned}
$$

> Beispiel:
>
> | B = 10 | B = 2 |
> |---|---|
> | $9.96 \cdot 10^1 + 5.0 \cdot 10^{-1}$ | $1.11 \cdot 2^1 + 1.0 \cdot 2^{-1}$ |
> | $= 9.96 \cdot 10^1 + 0.05 \cdot 10^1$ | $= 1.11 \cdot 2^1 + 0.01 \cdot 2^1$ |
> | $= 10.01 \cdot 10^1$ | $= 10.0 \cdot 2^1$ |
> | $= 1.001 \cdot 10^2$ | $= 1.0 \cdot 2^2$ |

Subtraktion für $r_1 \geqslant r_2$ (alle Rechenschritte in der Basis B):

$$
\begin{aligned}
& r_1 - r_2 \\
&= m_1 \cdot B^{e_1} - (m_2 \cdot B^{e_2 - e_1}) \cdot B^{e_1} && \text{(Exponentenangleich)} \\
&= (m_1 - m_2 \cdot B^{e_2 - e_1}) \cdot B^{e_1} && \text{(Subtraktion der Mantissen)} \\
&= m \cdot B^{e} && \text{(Normierung - falls nötig)}
\end{aligned}
$$

Beispiel:

B = 10	B = 16
$1.6 \cdot 10^2 - 7.1 \cdot 10^1$	$1.9 \cdot 16^2 - A.A \cdot 16^1$
$= 1.6 \cdot 10^2 - 0.71 \cdot 10^2$	$= 1.9 \cdot 16^2 - 0.AA \cdot 16^2$
$= 0.89 \cdot 10^2$	$= 0.E6 \cdot 16^2$
$= 8.9 \cdot 10^1$	$= E.6 \cdot 16^1$

Multiplikation/Division

Die Addition/Subtraktion von Zahlen in normierter Gleitkomma-Darstellung läuft in folgenden Schritten ab:

1. Multiplikation/Division der beiden Mantissen: muss gleichzeitig mit dem nächsten Schritt erfolgen
2. Addition/Subtraktion der Exponenten: Addition bei Multiplikation, Subtraktion bei Division.
3. Normierung (falls nötig): Bringe das Ergebnis wieder in die normierte Gleitkomma-Darstellung mit einem Exponenten $1 \leqslant |m| < B$, falls es nicht schon so da steht.

Multiplikation (alle Rechenschritte in der Basis B):

$$
\begin{aligned}
& r_1 \cdot r_2 \\
&= (m_1 \cdot m_2) \cdot B^{e_1 + e_2} && \text{(Exponenten addieren)} \\
&= m \cdot B^{e} && \text{(Normierung - falls nötig)}
\end{aligned}
$$

Beispiel:

B = 10	B = 2
$(6.0 \cdot 10^0) \cdot (2.5 \cdot 10^1)$	$(1.1 \cdot 2^0) \cdot (1.1 \cdot 2^1)$
$= 15 \cdot 10^1$	$= 10.01 \cdot 2^1$
$= 1.5 \cdot 10^2$	$= 1.001 \cdot 2^2$

Beispiel: Multiplikation in der Basis B

- $(6.0)_{10} \cdot (2.5)_{10}$
  $= (6.0)_{10} \cdot (2.0)_{10} + (6.0)_{10} \cdot (0.5)_{10}$
  $= (12.0)_{10} + (3.0)_{10} = (15.0)_{10}$
- $(1.1)_2 \cdot (1.1)_2$
  $= (1.1)_2 \cdot (1.0)_2 + (1.1)_2 \cdot (0.1)_2$
  $= (1.1)_2 + (0.11)_2 = (10.01)_2$

Division (alle Rechenschritte in der Basis B):

$$
\begin{aligned}
& r_1 / r_2 \\
&= (m_1 / m_2) \cdot B^{e_1 - e_2} && \text{(Exponenten subtrahieren)} \\
&= m \cdot B^{e} && \text{(Normierung - falls nötig)}
\end{aligned}
$$

Beispiel:

$$
\begin{array}{c}
B = 8 \\
\hline
(2.4 \cdot 8^0) / (5.0 \cdot 8^2) \\
= 0.4 \cdot 8^{-2} \\
= 4.0 \cdot 8^{-1}
\end{array}
$$

$$
\begin{array}{c}
\text{Vielfache von 5 zur Basis } B = 8 \\
\hline
(1)_8 \cdot (5)_8 = (5)_8 \\
(2)_8 \cdot (5)_8 = (12)_8 \\
(3)_8 \cdot (5)_8 = (17)_8 \\
(4)_8 \cdot (5)_8 = (24)_8
\end{array}
$$

3.5 Literaturverzeichnis

siehe Foliensatz