Digitale Rechenanlagen

MARIÁN VAJTERŠIC

Fachbereich Computerwissenschaften
Universität Salzburg
marian@cosy.sbg.ac.at
Tel.: 8044-6344

28. September 2017

Kapitel 2: Zahlendarstellung

Erste Digitalrechenanlagen wurden entwickelt, um einfache numerische Berechnungen durchzuführen.
[Heutzutage löst diese Aufgaben jeder Taschenrechner (sogar noch viel mehr) in wesentlich kürzerer Zeit.]

Das zentrale Problem beim Rechnen mit Computern:
Wie sollen Zahlen dargestellt werden?

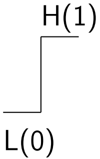
Wir wissen schon:
Die Repräsentation von Information ist gleichbedeutend mit der Codierung von Zeichen.

Es gibt aber folgende (hardwaremäßige) Einschränkung für die Repräsentation von Zahlen in Digitalrechenanlagen:

- Der Code für die Darstellung von Zahlen muss eine feste Länge haben!

Im Alltag benutzt man beim Rechnen fast immer Dezimalzahlen [d. h. diese Zahlen sind durch die Zeichen $0, 1, \ldots, 9$ codiert], aber in Digitalrechenanlagen gibt es nur zwei Zustände:

$\underline{0}$, $\underline{1}$ $\longrightarrow$ Spannungspegel



Also Zahlen werden in Digitalrechenanlagen als Folge fester Länge von **0**,**1**-Zeichen dargestellt.

Die Aufgabe ist daher:
Eine Codierung $C : D \to B^n$ finden, die eine Untermenge D der Dezimalzahlen (Wörter) aus dem Alphabet $A = \{0, 1, ..., 9\}$ auf B^n abbildet [d. h. man versucht, Dezimalzahlen binär mit n Stellen darzustellen].

Beispiel:

$$\begin{array}{ccc} 0 & \to & 1011 \\ 1 & \to & 1100 \\ \ldots & & \\ 13 & \to & 1000 \end{array}$$

Jeder der Dezimalzahlen [aus der angegebenen Untermenge] wird ein eindeutiger Wert zugeordnet.

Mit Binärwörtern der Länge $\boldsymbol{n}$ können wir maximal $2^{\boldsymbol{n}}$ verschiedene Dezimalzahlen codieren.

Prinzipiell könnten wir jeder Zahl aus der Untermenge von $2^{\underline{n}}$ Dezimalzahlen eindeutig eine binäre $\underline{n}$-Bit Zahl zuordnen, also eine tabellarische Zuordnung erzeugen.
Diese wäre ineffizient, da wir dann zur Decodierung eine Tabelle mit 2^n Einträgen bräuchten!

Anstelle einer expliziten (tabellarischen) Codierung wird eine analytische Methode zur Konstruktion eines Binärcodes für Dezimalzahlen benutzt.

Einfache Binärdarstellung

[Darstellung einer Dezimalzahl im polyadischen Zahlensystem]

Die analytische Methode zur (De-)Codierung von Dezimalzahlen basiert auf folgender Definition:

Definition

[Darstellung einer Dezimalzahl im polyadischen Zahlensystem] In einem polyadischen Zahlensystem [d. h. die Basis kann mehrere Werte annehmen] mit der Basis $B \in \mathbb{N},\ B > 1$ wird eine Dezimalzahl $m \in \mathbb{N}$ als Summe von gewichteten Potenzen von B folgendermaßen dargestellt:

$$(1) \quad m = \sum_{i=0}^{n-1} b_i B^i = b_0 B^0 + b_1 B^1 + b_2 B^2 + \ldots + b_{n-1} B^{n-1} = \\ = b_0 + b_1 B + b_2 B^2 + \ldots + b_{n-1} B^{n-1}.$$

[Darstellung einer Dezimalzahl im polyadischen Zahlensystem]

Die Gewichte b_i sind die Ziffern des Zahlensystems zur Basis B $[b_i \in \{0, 1, ..., B - 1\}]$, wobei die Ziffer b_0 ganz rechts steht und die Zahl dann lautet: $b_{n-1} b_{n-2} \dots b_1 b_0$.

Für uns ist das **dyadische** Zahlensystem (**Binärsystem**), aufgrund der Anwendung in Digitalrechenanlagen, von spezieller Bedeutung.

Beispiel: Einige wichtige Zahlensysteme

Zahlensystem	Basis	Ziffern
Binär	2	0,1
Oktal	8	0,1,2,3,4,5,6,7
Dezimal	10	0,1,2,3,4,5,6,7,8,9
Hexadezimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

[Darstellung einer Dezimalzahl im polyadischen Zahlensystem]

Um Verwechslungen zu vermeiden [z. B. kann 9 in einer Dezimalzahl, sowie auch in einer Hexadezimalzahl als Ziffer auftreten] wird das verwendete Zahlensystem [wenn nötig] als Index angegeben:

$1000_{(10)}$: Tausend im Dezimalsystem
$1000_{(2)}$: Eine „andere“ Zahl im Binärsystem

Beispiel: Zahlenumwandlung ins Dezimalsystem

$$127_{(10)} = 7 \cdot \underbrace{10^0}_{B^0} + 2 \cdot \underbrace{10^1}_{B^1} + 1 \cdot \underbrace{10^2}_{B^2} = 127_{(10)}$$

$$10101_{(2)} = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 21_{(10)}$$

$$3147_{(8)} = 7 \cdot 8^0 + 4 \cdot 8^1 + 1 \cdot 8^2 + 3 \cdot 8^3 = 1639_{(10)}$$

$$1A2C_{(16)} = 12 \cdot 16^0 + 2 \cdot 16^1 + 10 \cdot 16^2 + 1 \cdot 16^3 = 6700_{(10)}$$

Zahlenumwandlung

[Übergang zwischen Zahlensystemen]

Wie kann man Dezimalzahlen in ein anderes Zahlensystem umwandeln?

$\rightarrow$ Sukzessive Division

[Die Gleichung (1) wird durch die Basis B dividiert.]

$$\frac{m - b_0}{B} = b_1 + b_2 B + \ldots + b_{n-1} B^{n-2} = m'$$

$\rightarrow m = m'B + b_0 \rightarrow$ Die Ziffer b_0 ist der Rest bei der Division von m durch B.

Mit m' verfährt man in gleicher Weise. Der nächste Rest ist die Ziffer b_1, usw., bis $m' = 0$.

Zahlenumwandlung

[Übergang zwischen Zahlensystemen]

Beispiel: (Sukzessive Division) $47_{(10)} \stackrel{?}{=} x_{(2)} \rightarrow B = \underline{2}$

m		m'		b_i	
$47:\underline{2}$	$=$	23	(+Rest	$\underline{1}$)	$\underline{b_0}$
23		11		$\underline{1}$	$\underline{b_1}$
11		5		$\underline{1}$	$\underline{b_2}$
5		2		$\underline{1}$	$\underline{b_3}$
2		1		$\underline{0}$	$\underline{b_4}$
1		0		$\underline{1}$	$\underline{b_5}$

$$
\begin{array}{rcl}
47_{(10)} & = & \begin{array}{cccccc} 1 & 0 & 1 & 1 & 1 & 1_{(2)} \\ b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \end{array} \\
& = & 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^5 \quad = \\
& = & 1 + 2 + 4 + 8 + 32 \quad = \quad 47_{(10)}
\end{array}
$$

Zahlenumwandlung

[Wie groß ist die größte darstellbare Zahl zur Basis B mit n Stellen?]

$$m_{max} = b_{max} \sum_{i=0}^{n-1} B^i = b_{max}(1 + B + B^2 + \ldots + B^{n-1}) .$$

$(1 + B + \ldots + B^{n-1})$: Geometrische Reihe mit Quotient q:

$$\sum_{i=0}^{n-1} q^i = \frac{1 - q^n}{1 - q} .$$

Zahlenumwandlung

[Wie groß ist die größte darstellbare Zahl zur Basis B mit n Stellen?]

In unserem Fall: $q = B$, $b_{max} = \underline{B - 1}$

[Da ein Zahlensystem zur Basis B aus den Ziffern von $0, 1, 2, \ldots, B - 1$ besteht $\to$ die größte Ziffer ist $B - 1$.]

Dann:

$$\underline{m_{max}} = b_{max} \sum_{i=0}^{n-1} B^i = (B-1)\frac{1-B^n}{1-B} = \underline{B^n - 1}\,.$$

[Anders: Die größte darstellbare Zahl zur Basis B mit n Stellen ist $B^n - 1$, da B^n nicht mehr dargestellt werden kann. $B = 2 \to 2^n - 1$]

Darstellung negativer Zahlen

Negative Zahlen

[Darstellung negativer Zahlen]

Bisher haben wir nur positive [ganze] Zahlen betrachtet.

Nun: Betrachten die [binäre] Darstellung negativer Zahlen.

Codierung negativer Zahlen

Wie codiert man negative Zahlen?
Durch Voranstellen eines Minuszeichens:

$$-17 = (-1) \cdot 17 = \ominus\, 17.$$

$\downarrow$
Minus-Zeichen verwendet

Wie kann man das bei Binärzahlen machen?

$\ominus\, 101$

$\downarrow$
Das Minuszeichen muss integriert werden.
[d. h. auch mit 0/1 darstellbar sein.]

Naive Codierung des Minuszeichens

- MSB: Most Significant Bit b_{n-1} [Die wichtigste Stelle: ganz links]
- LSB: Least Significant Bit b_0 [ganz rechts]

$$\text{Wenn} \quad \begin{array}{l} \text{MSB=0: positive Zahl} \\ \text{MSB=1: negative Zahl.} \end{array}$$

Führt zu einem Problem:

$$(\underline{-}1)_{(10)} + (\underline{-}1)_{(10)} = \underline{1}01_{(2)} \oplus \underline{1}01_{(2)} = \underline{0}10_{(2)} = \underline{+}2_{(10)}$$

$$\begin{array}{rl} 101 & [-1] \\ 101 & [-1] \\ {\scriptstyle (1)\;\;(1)} & \\ \hline (1)|010 & [+2] \end{array}$$

Problem: Definierte Anzahl der Stellen.

Verbesserung: Geeignete Wahl des MSB.

Die Idee

Ein Vorzeichenbit nicht nur zur Indikation des Vorzeichens zu verwenden sondern auch zur Darstellung der Zahlen.

Wir wissen: Die größte darstellbare [natürliche] Binärzahl mit $\boldsymbol{n}$ Stellen ist $\mathbf{2^n - 1}$.
Wir verwenden diese $\boldsymbol{n}$ Stellen für die Darstellung von Zahlen aus dem Intervall $[-2^{n-1}+1,\, 2^{n-1}-1]$, d. h. insgesamt $\mathbf{2^n - 1}$ Zahlen mit 2 Null-Darstellungen
und
interpretieren das MSB [hier das Bit links außen] als Vorzeichenbit.

Für einfache Addition [Subtraktion] von Zahlen in dieser Darstellung muss man das Gewicht des MSB richtig wählen. Eine Möglichkeit ist die **1-Komplement-Darstellung**.

1-Komplement-Darstellung

Die **1-Komplement-Darstellung** einer Zahl $z \in \mathbb{Z}$ ist folgendermaßen definiert:

$$z = -b_{n-1}(2^{n-1} - 1) + \sum_{k=0}^{n-2} b_k 2^k \,.$$

Wenn $b_{n-1} = 1$ [d. h. MSB ist gesetzt], dann ist z negativ [-0 inklusive].

[Da die größte darstellbare Zahl mit $n-1$ Zeichen 2^{n-1} ist, gilt $\sum_{k=0}^{n-2} b_k 2^k \leq 2^{n-1} - 1$.]

Wenn $b_{n-1} = 0$ [d. h. MSB ist nicht gesetzt], dann fällt der negative Teil weg und z ist positiv [$+0$ inklusive].

1-Komplement-Darstellung

Also, wenn

1. $b_{n-1} = 1 \quad : \quad -2^{n-1} + 1 \leq z \leq 0$
2. $b_{n-1} = 0 \quad : \quad 0 \leq z \leq 2^{n-1} - 1\,.$

Darstellung von z:

1. Für z sucht man das positive z' sodass $z' = z + (2^{n-1} - 1)$ und stellt dieses in einfacher Binärdarstellung mit $b_{n-1} = 1$ [negatives Vorzeichen] dar.
2. Für z ergibt sich die einfache binäre Darstellung mit $b_{n-1} = 0$ [positives Vorzeichen].

Bemerkung:

Für $z = 0$ ergeben sich zwei Darstellungen:

1. $z = -0$
2. $z = +0$

1-Komplement-Darstellung

Beispiel: [8-Bit-Zahlen im 1-Komplement]

- $z_{(10)} = 1 \quad \rightarrow$ Fall 2. $z'_{(10)}$ wird nicht berechnet.

$$k_1(1)_{(2)} = \underbrace{0}_{\text{Vorzeichenbit}}0000001$$

- $z_{(10)} = -1 \rightarrow$ Fall 1. $z'_{(10)}$ wird berechnet:

$$z'_{10} = \overbrace{-1 + (2^{(8-1)} - 1)}^{z_{(10)}+(2^{n-1}-1)} = -1 + (2^7 - 1) = -1 + 127 = 126$$

dann einfach binär codiert [$126_{(10)} = 1111110_{(2)}$] und das negative Vorzeichen hinzugefügt.

$$k_1(-1)_{(2)} = \underbrace{1}_{\text{Vorzeichenbit}}1111110$$

1. Darstellung der Zahl inklusive Vorzeichenbit = 1.
2. Einfache Codierung inklusive Vorzeichenbit = 0.

1-Komplement-Darstellung

Beispiel: [8-Bit-Zahlen im 1-Komplement]

- $z_{(10)} = 127 \quad \rightarrow$ Fall 2. $\quad z'_{(10)}$ wird nicht berechnet.
  $$k_1(127)_{(2)} = \underline{0}1111111$$
- $z_{(10)} = -127 \rightarrow$ Fall 1. $\quad z'_{(10)} = -127 + 127 = 0.$
  $$k_1(-127)_{(2)} = \underline{1}0000000$$

1-Komplement-Darstellung

Beispiel: [Zwei 8-Bit-Zahlen im 1-Komplement]

- $z_{(10)} = 0 \quad \to$ Fall 2.

$$k_1(0)_{(2)} = \underline{0}0000000$$

- $z_{(10)} = -0 \to$ Fall 1. $\quad z'_{(10)} = -0 + 127 = 127 \xrightarrow{1.}$

$$k_1(-0)_{(2)} = \underline{1}1111111$$

Bemerkung:
Es ist ersichtlich, dass für $k_1(z)_{(2)} = b_{n-1}b_{n-2}...b_0$ gilt:
$k_1(-z)_{(2)} = \neg b_{n-1}\neg b_{n-2}...\neg b_0$,
wobei $\neg 0 = 1$ und $\neg 1 = 0$, also das binäre Komplement darstellt.

1-Komplement-Darstellung

Bemerkung: Für eine Zahl $z_{(1)}$ im 1-er Komplement gilt: $\overline{z}_{(1)} = -z_{(1)}$.

Beweis.

Zu zeigen ist, dass $\overline{z}_{(1)} + z_{(1)} = 0$.

$$\overline{z}_{(1)} = -\overline{b_{n-1}}(2^{n-1} - 1) + \sum_{k=0}^{n-2} \overline{b_k} 2^k$$

$$z_{(1)} = -b_{n-1}(2^{n-1} - 1) + \sum_{k=0}^{n-2} b_k 2^k$$

$$\overline{z}_{(1)} + z_{(1)} = -(\underbrace{\overline{b_{n-1}} + b_{n-1}}_{1})(2^{n-1} - 1) + \sum_{k=0}^{n-2} (\underbrace{\overline{b_k} + b_k}_{1}) 2^k$$

$$= -1(2^{n-1} - 1) + \sum_{k=0}^{n-2} 2^k = -(2^{n-1} - 1) + (2^{n-1} - 1) = 0 .$$

□

1-Komplement-Darstellung

Beispiel:

$$z_{(1)} = 0001 = \underline{1} \qquad \overline{z}_{(1)} = \overline{0001} = 1110 = -7 + 4 + 2 = -\underline{1}$$
$$z_{(1)} = 1011 = -\underline{4} \qquad \overline{z}_{(1)} = \overline{1011} = 0100 = \underline{4}$$
$$z_{(1)} = 1111 = -\underline{0} \qquad \overline{z}_{(1)} = \overline{1111} = 0000 = +\underline{0}$$

Addition im 1-Komplement

Wenn man addieren kann, kann man auch subtrahieren.
[Da die Subtraktion $a - b$ die Rückführung auf die Addition $a + (-b)$ erlaubt.]

Zu beachten: Das Bit b_n [links vom MSB] kann bei einer Addition verändert werden!

Bei der Addition im 1-Komplement müssen wir dieses Bit [im Falle von $b_n = 1$] an der Stelle b_0 hinzuaddieren [Einserrücklauf].

Es entsteht also ein Übertrag [eine $n + 1$ -stellige Zahl]. Dieser wird zu den ursprünglichen Zahlen mit jeweils verdoppeltem MSB [b_{n-1}] addiert.

Wenn dabei $b_n \neq b_{n-1}$ ist, dann kommt es zu einem Überlauf [Overflow], d. h. die Summe wird die maximal darstellbare Zahl überschreiten → Falsches Ergebnis!

Beispiel: $[-0 \oplus -0]$

$$
\begin{array}{lcccccccccccl}
\text{-0} & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
\text{-0} & \oplus & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \text{Übertrag} \\
\hline
& & & \underbrace{1}_{b_n} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 &
\end{array}
$$

$$
\begin{array}{lccccccccccccl}
\text{-0} & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \text{Die Summanden} \\
\text{-0} & \oplus & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \text{mit doppeltem MSB} \\
& \oplus & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & [b_n \text{ an der Stelle } b_0] \\
\hline
& & \underbrace{1}_{b_{n+1}} & \underbrace{1}_{b_n} & \underbrace{1}_{b_{n-1}} & 1 & 1 & 1 & 1 & 1 & 1 & 1 &
\end{array}
$$

b_{n+1} ist uninteressant.
$b_n = b_{n-1} \rightarrow$ es ist kein Überlauf aufgetreten.

$\rightarrow$ Das Ergebnis ist ok.

Für **das Ergebnis** sind nur die Stellen $\underline{b_{n-1} \ \ldots \ b_0}$ relevant!
$\rightarrow -0 \oplus -0 = 11111111 = -0$

Beispiel: [100 $\oplus$ 28: 8-Bit-Darstellung]

$100_{(10)} + 28_{(10)} = 128_{(10)}$
Diese Zahl ist außerhalb des Intervalls [-127, 127] [die Zahlen, die durch 8 Bit-Stellen im 1-Komplement noch darstellbar sind].

$$
\begin{array}{lccccccccccc}
100 & & & & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
28 & \oplus & & & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
 & & \underbrace{0}_{b_n} & & \underbrace{1}_{b_{n-1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

$\rightarrow b_n \neq b_{n-1}$: Überlauf! $[010000000 = -127 + 0 = -127_{(1K)} \neq 128!]$

Hier ist $b_n = 0$, also müssen wir den Übertrag nicht zu den Summanden [mit verdoppeltem MSB] addieren – es würde keine Veränderung des Resultats bringen.

Beispiel: $[-77 \oplus -52]$

$$
\begin{array}{lcccccccccc}
\text{-77} & & & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
\text{-52} & \oplus & & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
\hline
& & \underbrace{1}_{b_n} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1
\end{array}
\quad \text{Übertrag}
$$

$$
\begin{array}{lccccccccccc}
\text{-77} & & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \\
\text{-52} & \oplus & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & \\
& \oplus & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \\
\hline
& \underbrace{1}_{b_{n+1}} & \underbrace{1}_{b_n} & \underbrace{0}_{b_{n-1}} & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

Die Summanden mit doppeltem MSB
$[b_n$ an der Stelle $b_0]$

b_{n+1} ist uninteressant.
$b_n \neq b_{n-1} \rightarrow$ Überlauf

$\rightarrow$ Das Ergebnis ist <u>ungültig</u> (und falsch).

$01111110 = 64 + 32 + 16 + 8 + 4 + 2 = 126 \neq -129!$

Überlauf

Überlauf: Feststellung mittels Vergleich von b_n mit b_{n-1}. Das Ergebnis dieses Vergleichs wird als Flag [Bit] in der CPU in einem Statusregister angeführt.

Statusregister 0/1 $\leftarrow$

Overflow Flag						

Ist das Flag $= 1$, dann ist ein Überlauf aufgetreten. Das Steuerprogramm kann so einfach erkennen, ob das Resultat gültig ist oder nicht.

Nachteile der 1-Komplement Darstellung

Die Nachteile der 1-Komplement Darstellung sind:

- der Einserrücklauf
- die doppelte Darstellung der 0

Diese beiden Nachteile werden eliminiert durch die **2-Komplement-Darstellung**.

2-Komplement-Darstellung

Für eine Zahl $z \in \mathbb{Z}$ gilt:

$$z = -b_{n-1}2^{n-1} + \sum_{k=0}^{n-2} b_k 2^k \,.$$

Die 0 hat nur eine Darstellung: **0**0000000 (für $n = 8$).

[Es muss $b_{n-1} = 0$ sein $\rightarrow b_{n-2} = \ \ldots \ = b_0 = 0$, denn wenn $b_{n-1} = 1$
$\rightarrow -2^{n-1} + (2^{n-1} - 1) = -1 < 0$,
weil $2^{n-1} - 1$ die größte darstellbare positive Zahl mit $n - 1$ Bits ist.]

Das ermöglicht die Darstellung einer zusätzlichen Zahl

$$\begin{aligned} \rightarrow \qquad & z \in [-2^{n-1}, 2^{n-1} - 1] \\ n = 8: \quad & z \in [-128, 127]\,. \end{aligned}$$

2-Komplement-Darstellung

Darstellung von z: Wir unterscheiden zwei Intervalle (Analogie zum 1-Komplement):

1. $b_{n-1} = 1 \quad : \quad -2^{n-1} \leq z < 0$
2. $b_{n-1} = 0 \quad : \quad 0 \leq z \leq 2^{n-1} - 1$

1. Für z sucht man das positive $z' = z + 2^{n-1}$ und stellt dieses in einfacher Binärdarstellung **mit** $b_{n-1} = 1$ [negatives Vorzeichen] dar.
2. Hier wird z in einfacher binärer Darstellung **mit** $b_{n-1} = 0$ [positives Vorzeichen] dargestellt.

2-Komplement-Darstellung

Beispiel: [8-Bit-Zahlen im 2-Komplement]

$z_{(10)}$	$z'_{(10)}$	$k_2(z)_{(2)}$	
1	-	00000001	
-1	$\underline{127}$	11111111	$\rightarrow z' = z + 2^{n-1} = -1 + 128 = \underline{127}$ (Vorzeichenbit: 1)
127	-	01111111	
-127	$\underline{1}$	10000001	$\rightarrow z' = z + 2^{n-1} = -127 + 128 = \underline{1}$ (Vorzeichenbit: 1)
-128	0	10000000	

2-Komplement-Darstellung

Bildungsvorschrift für negative Zahlen im 2-Komplement:

$$
\begin{array}{llcccccc}
\text{wenn} & k_2(z)_{(2)} & = & & b_{n-1} & b_{n-2} & \dots & b_0 & \rightarrow \\
& k_2(-z)_{(2)} & = & & \neg b_{n-1} & \neg b_{n-2} & \dots & \neg b_0 & \\
& & & \oplus & 0 & 0 & \dots & 1 &
\end{array}
$$

Beispiel:

$$
\begin{array}{rcccccccccc}
127 & = & & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\\
\text{-127} & = & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& & \oplus & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
& & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

Addition im 2-Komplement

Hier wird der Einserrücklauf schon bei der Komplementbildung berücksichtigt
$\rightarrow$ die Addition/Subtraktion wird vereinfacht.

Vorgang:
Verdopplung von MSB [b_{n-1}] der beiden Summanden, danach wird die Addition durchgeführt.

Falls $b_n \neq b_{n-1}$: Überlauf aufgetreten.

Tritt kein Überlauf auf [also $b_n = b_{n-1}$], so ist das Ergebnis direkt an den Stellen der Summe $b_{n-1} ... b_1 b_0$ abzulesen [b_{n+1} bleibt unberücksichtigt].

Beispiel: Addition im 2-Komplement

$17_{(10)} + (-2)_{(10)} = 17_{(10)} - 2_{(10)} = 15_{(10)}$

$$\begin{array}{rcccccccccl} 2_{(10)} & : & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\ -2_{(10)} & : & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & [z' = -z + 128 = 126] \end{array}$$

Verdopplung und Addition:

$$\begin{array}{llccccccccccc} 17_{(10)} & & & \underline{0} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ -2_{(10)} & \oplus & & \underline{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline & & \underbrace{1}_{b_{n+1}} & \underbrace{0}_{b_n} & \underbrace{0}_{b_{n-1}} & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

$b_n = b_{n-1} = 0 \rightarrow$ kein Überlauf, b_{n+1} unberücksichtigt

Das Vorzeichen ist positiv $(\underline{b_{n-1}} = 0)$ und $(\underline{b_{n-2}b_{n-3}...b_0})_{(2)} = \underline{0001111}_{(2)}$.

$\rightarrow$ Das Ergebnis ist $0\underline{0001111}_{(2)} = 15_{(10)}$.

Addition im 1- und 2-Komplement: Zusammenfassung

[1-Komplement]

- Die Summanden im 1-Komplement darstellen
- Addition der Summanden [beide $\underline{n}$ Bit lang]

$$
\begin{array}{c|cccc}
\underline{n} & n-1 & & & 0 \\
\downarrow & \mathrm{x} & & \ldots\ldots & . \\
\downarrow & \mathrm{x} & & \ldots\ldots & . \\
\hline
\bullet & . & & \ldots\ldots & .
\end{array}
$$

  - a) $\bullet = 1$: Übertrag
  - b) $\bullet = 0$: kein Übertrag

Addition im 1- und 2-Komplement: Zusammenfassung

[1-Komplement]

- a) Übertrag: Verdopplung des Vorzeichenbits $[n-1]$ [die Summanden sind dann $n+1$ Bits lang], Addition von $0\ldots01$ [aus dem Übertrag] zu den Summanden.

$$
\begin{array}{ccccc}
n+1 & \underline{n} & \underline{n-1} & & 0 \\
 & x & x & \ldots & . \\
 & x & x & \ldots & . \\
 & 0 & 0 & \ldots & 1 \\
\hline
\bullet & \odot & \otimes & \ldots & .
\end{array}
$$

  $\bullet$ [Position $n+1$] belanglos.
  Wenn $\odot = \otimes$ [Position n und $n-1$]: ok, kein Überlauf aufgetreten.
  Wenn $\odot \neq \otimes$: Überlauf, Ergebnis ist ungültig.
  Ergebnis: $\otimes \ldots\ldots$ [also eine n Bit lange Zahl].

- b) Kein Übertrag: Wie a), aber ohne Einserrücklauf-Phase [also gleich die Summanden mit verdoppeltem Vorzeichenbit addieren]

Probe: $(Ergebnis)_{(10)} = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \ldots + b_{n-2} \cdot 2^{n-2} - b_{n-1} \cdot (2^{n-1} - 1)$

Addition im 1- und 2-Komplement: Zusammenfassung

[2-Komplement]

- Die Summanden im 2-Komplement darstellen
- Addition wie nach b) (vorige Folie)

Probe: $(Ergebnis)_{(10)} = b_0 \cdot 2^0 + b_1 \cdot 2^1 + ... + b_{n-2} \cdot 2^{n-2} - b_{n-1} \cdot 2^{\underline{n-1}}$

Multiplikation von Binärzahlen

Wir können auf die bekannten Methoden für das Dezimalsystem zurückgreifen.

- Klassisches Verfahren:
  Die Multiplikation wird mit den einzelnen Zahlen des Multiplikators ausgeführt und die Einzelergebnisse entsprechend ihrer Wertigkeit [Position] addiert.

Multiplikation von Binärzahlen

Beispiel: Binärmultiplikation

Das Produkt $7_{(10)} \cdot 3_{(10)}$ in 4-Bit Binärdarstellung:

$7_{(10)} \cdot 3_{(10)} =$

$$
\begin{array}{cccccccccccc}
 & 0 & 1 & 1 & 1 & & \otimes & \underline{0} & 0 & 1 & \underline{1} & \\
\hline
\circledcirc & 0 & 0 & 0 & 0 & & & & & & & (\,\underline{0}\cdot 0111) \\
 & & 0 & 0 & 0 & 0 & & & & & & \\
 & & & 0 & 1 & 1 & 1 & & & & & \\
\circledast & & & & 0 & 1 & 1 & 1 & & & & (\,\underline{1}\cdot 0111) \\
\hline
= 21_{(10)}: & & & 1 & 0 & 1 & 0 & 1 & & & &
\end{array}
$$

Also: Die Binärmultiplikation ist auf mehrere Additionen zurückzuführen, weil bei den einzelnen Multiplikationen nur zwei Teilergebnisse möglich sind - der Multiplikand ($\circledast$) oder die 0 ($\circledcirc$). Diese Eigenschaft wird bei der Konstruktion von Multiplizierern [Hardware für die Multiplikation] ausgenützt.

Multiplikation von Binärzahlen

Problem für die Hardware:

Die Anzahl der Stellen des Produkts ist $p = d + k$ $[5 = 3 + 2]$.

[d: Stellenanzahl des Multiplikanden
k : des Multiplikators
(0-Stellen am Anfang nicht gezählt)]

Multiplikation von Binärzahlen

[Ägyptische Multiplikation]

- Ägyptische Multiplikation:

Vorgang:

- a) man schreibt die beiden zu multiplizierenden Zahlen nebeneinander
- b) verdoppelt in jeder neuen Zeile die linke Zahl
- c) halbiert die rechte Zahl [ganzzahlig]
- d) streicht jene Zeilen, in denen die rechte Zahl gerade ist
- e) addiert die verbleibenden <u>*linken*</u> Zahlen

Beispiel:

a)	22		21	
b)	~~44~~	c)	~~10~~	d)
b)	88	c)	5	
b)	~~176~~	c)	~~2~~	d)
b)	352	c)	1	
e)	462			

Multiplikation von Binärzahlen

[Ägyptische Multiplikation, Eigenschaften der Methode]

Neben der Addition wird nur die Verdopplung und Halbierung der Zahlen benötigt. Diese beiden Operationen lassen sich aber mit Binärzahlen sehr einfach realisieren.

- Verdopplung: eine Verschiebung um eine Stelle nach links [shift left, SHL]
- Halbierung: eine Verschiebung um eine Stelle nach rechts [shift right, SHR]

Multiplikation von Binärzahlen
[Halbierung/Verdopplung]

Beispiel:

$$
\begin{aligned}
27 \cdot 2 \quad &= \mathrm{SHL}(\underline{11011}) &&= \underline{11011}0 &&= 54 \\
33 \;/\; 2 \quad &= \mathrm{SHR}(\underline{10000}1) &&= \underline{10000} &&= 16
\end{aligned}
$$

[Die freiwerdende Stelle wird mit dem Bit 0 belegt.]

Bemerkung: Diese beiden Operationen gehören zu Befehlen einer gängigen Maschinensprache [Assemblerprogrammierung].

Multiplikation zur Basis $B \neq 2, 10$

- $23_{(8)} \cdot 16_{(8)}$

$$\begin{array}{l} 22_{(8)} \\ 14_{(8)} \\ \underline{23_{(8)}} \\ {\color{red}4}{\color{blue}1}2_{(8)} = 4 \cdot 8^2 + 1 \cdot 8^1 + 2 \cdot 8^0 = 266_{(10)} \end{array}$$

$6 \cdot 3 = 18 = 2 \cdot 8 + 2 = 22_{(8)}$
$6 \cdot 2 = 12 = 1 \cdot 8 + 4 = 14_{(8)}$
$1 \cdot 23_{(8)} = 23_{(8)}$

${\color{blue}1}$: $3 + 4 + 2 = 9 = {\color{green}11}_{(8)}$
${\color{red}4}$: ${\color{green}1} + 2 + 1 = 4 = 4_{(8)}$

Probe:
$23_{(8)} = 19_{(10)}$
$16_{(8)} = 14_{(10)}$
$19 \cdot 14 = 266_{(10)}$

Multiplikation zur Basis $B \neq 2, 10$

- $41_9 \cdot 37_9$

$$
\begin{array}{l|l}
\;\;\underbrace{31}\underline{7}_9 & 7 \cdot 1 = 7 = \underline{7}_9 \\
133_9 & 7 \cdot 4 = 28 = \underbrace{31}_9 \\
\hline
\end{array}
$$

$$1647_9 = 1 \cdot 9^3 + 6 \cdot 9^2 + 4 \cdot 9 + 7 = 729 + 486 + 36 + 7 = 1258_{(10)}$$

$$
\begin{aligned}
\text{Probe: } 41_9 &= 37_{10} \\
37_9 &= 34_{10} \\
\\
37 \cdot 34 &= 1258_{10}
\end{aligned}
$$

Fazit: Multiplikation wie üblich
aber

- Die Darstellung von Teilprodukten zur Basis B
- Die Addition der Teilprodukte auch zur Basis B.

Division von Binärzahlen

- $111111_2 : 1001_2 = ?$

$$
\begin{array}{r}
111111_2 : 1001_2 = 111_2 \\
-1001 \\
\hline
01101 \\
-1001 \\
\hline
01001 \\
-1001 \\
\hline
0
\end{array}
$$

$$
\text{Probe:} \quad
\begin{array}{lclcr}
111111 & = & 2^6 - 1 & = & 63_{10} \\
1001 & = & & = & 9_{10}
\end{array}
$$

$$63 : 9 = 7_{(10)} = 111_{(2)}$$

Division von Binärzahlen

- $1001,11_2 : 1,101_2 = ?$

$$
\begin{array}{l}
1001{,}11_2 : 1{,}101_2 = \\
1001110_2 : 1101_2 = 110_2 \\
\underline{-1101} \\
\ 001101 \\
\ \ \underline{-1101} \\
\ \ \ \ 00000
\end{array}
$$

Probe:

$$
\begin{array}{lclcl}
1001{,}11_2 & = & 9 + 0{,}5 + 0{,}25 & = & 9{,}75_{(10)} \\
1{,}101_2 & = & 1 + 0{,}5 + 0{,}125 & = & 1{,}625_{(10)}
\end{array}
$$

$$
\begin{array}{l}
9{,}75 : 1{,}625 \\
9750 : 1625 = 6 = (110)_{(2)} \\
\underline{-9750} \\
\ \ \ \ \ 0
\end{array}
$$

Fazit: Division wie üblich mit Operationen binäre Multiplikation und binäre Subtraktion.

Division im Zahlensystem zur Basis B≠2,10

- $11011_{(3)} : 11_{(3)} = ?$

$$
\begin{array}{l}
11011_{(3)} : 11_{(3)} = 1001 \\
-11_3 \\
\overline{00011} \\
-11 \\
\overline{00}
\end{array}
$$

$$
\text{Probe:} \quad
\begin{array}{rcrcr}
11011_{(3)} & = & 1+3+27+81 & = & 112_{(10)} \\
11_{(3)} & = & 1+3 & = & 4_{(10)}
\end{array}
$$

$$112_{(10)} : 4_{(10)} = 28_{(10)} = 1001_{(3)}$$

Division im Zahlensystem zur Basis B≠2,10

- $1017_8 : 37_8 = ?$

$$
\begin{array}{l}
\underline{101}7_8 : 37_8 = \underline{2}1_8 \\
\underline{-76_8}* \\
\;\;0\underline{\underline{3}}7_8\square \\
\underline{-37_8} \\
\;\;\;00
\end{array}
\qquad
\text{Probe:} \quad
\begin{array}{rcr}
1017_8 & = & 7+8+512 = 512_{(10)} \\
37_8 & = & 31_{(10)}
\end{array}
$$

– Abschätzung für 2 : $101_8 : 37_8 = 10{,}1 : 3{,}7 \approx 10_8 : 4_8 = 8 : 4 = 21_8$

– * Erklärung für 76_8 : durch $2_8 \cdot 37_8$

$$
\begin{array}{rcccl}
2 \cdot 7 & = & 14 & = & 16_8 \\
2 \cdot 3 & = & 6 & = & \underline{6_8} \\
 & & & & 76_8
\end{array}
$$

– $\square$ Erklärung für 37_8: Durch Subtraktion von $101_8 - 76_8$.
(!Also im Oktalsystem: 6 und wie viel ist $11_8 = 9_{10}$? Antwort: $\underline{\underline{3}}$)

- $23144{,}3_{(5)} : 44{,}321_{(5)} = ?$

$$23144{,}3_{(5)} : 44{,}321_{(5)} = 232{,}4\ldots_{(5)}$$

$$
\begin{array}{l}
23144300_{(5)} : \underset{\smile}{44}321_{(5)} = \underline{232},4_{(5)} \\
-144142_{(5)} \\
\hline
\underline{0323010}_{(5)} \\
\quad -244013_{(5)} \\
\hline
\qquad 234420_{(5)} \\
\qquad -144142_{(5)} \\
\hline
\qquad\quad 402230 \\
\qquad -343334 \\
\hline
\qquad\qquad 3341
\end{array}
$$

Probe:

$$
\begin{aligned}
23144{,}3_{(5)} &= 0{,}6 + 4 + 4 \cdot 5 + 1 \cdot 25 + 3 \cdot 125 + 2 \cdot 625 = \\
&= 1674{,}6_{(10)} \\
44{,}321_{(5)} &= 2 \cdot 0{,}04 + 3 \cdot 0{,}2 + 4 + 4 \cdot 5 = \\
&= 24{,}68_{(10)}
\end{aligned}
$$

$$
\begin{aligned}
1674{,}6 : 24{,}68 \approx 67{,}85 \quad &= 2 + 3 \cdot 5 + 2 \cdot 25 + 4 \cdot 0{,}2 = \\
&= 67{,}8 = \underline{\underline{232{,}4_{(5)}}}
\end{aligned}
$$

-Abschätzung für 2 : $23_5 : 4{,}4_5 \approx 13_{10} : 4{,}8_{10} = 2$

-Erklärung * für $32301_{(5)}$

$$
\begin{array}{ccccccc}
2 & + & \underline{1} & = & 3 & & \\
4 & + & \underline{0} & = & 4 & & \\
1 & + & \underline{3} & = & 4 & & \\
4 & + & \underline{2} & = & 11_5 & & =6 \\
(4+1) & + & \underline{3} & = & 13_5 & & =8 \\
(1+1) & + & \underline{0} & = & 2 & &
\end{array}
$$

-Abschätzung für 3 : $32_5 : 4{,}4_5 \approx 17 : 4{,}8 = 3$

-Abschätzung für 2 : $23_5 : 4{,}4_5 \approx 13 : 4{,}8 = 2$

-Abschätzung für 4 : $40_5 : 4{,}4_5 \approx 20 : 4{,}8 = 4$

Fazit:
- Abschätzung aufgrund weniger (2 ?) Ziffern
- Multiplikation zur Basis B
- !Subtraktion zur Basis B

Darstellung rationaler Zahlen

Darstellung rationaler Zahlen

In numerischen Berechnungen ist die Verwendung rationaler Zahlen $q \in \mathbb{Q}$ [Kommazahlen] unumgänglich.
[Eine rationale Zahl ist durch einen Bruch einer ganzen Zahl [$\mathbb{Z}$] und einer natürlichen Zahl [$\mathbb{N}$] darstellbar.]

- natürliche Zahlen $\mathbb{N} = \{1, 2, \ldots\}$
- ganze Zahlen $\mathbb{Z} = \{\ldots - 2, -1\} \cup \{0\} \cup \mathbb{N} = \{\ldots - 2, -1, 0, 1, 2, \ldots\}$

Im Rechner können nur endliche Zahlen dargestellt werden. Eine Erweiterung des polyadischen Zahlensystem ermöglicht die Darstellung [gewisser] rationaler Zahlen:

$$q = \sum_{k=\underline{-m}}^{n-1} b_k B^k = b_{-m} B^{-m} + b_{-m+1} B^{-m+1} + \ldots + b_0 + \ldots + b_{n-1} B^{n-1}$$

Festpunktdarstellung

Jede rationale Zahl q lässt sich als Summe einer ganzen Zahl $u \in \mathbb{Z}$ und einer rationalen Zahl v $(1 > v \geq 0)$ mit $q = u + v$ angeben:

- $-\frac{4}{1} = (-4.0)_{(10)} = -4 + \underline{0.0} = u + \underline{v}$
- $-\frac{37}{10} = (-3.7)_{(10)} = -4 + \underline{0.3} = u + \underline{v}$
- $\frac{2625}{1000} = (2.625)_{(10)} = 2 + \underline{0.625} = u + \underline{v}$

Einfaches Darstellungsformat

Festpunktdarstellung

- $q_{(10)} = (u_n \underbrace{u_{n-1} \ldots u_0}_{u} \odot \underbrace{v_{-1} \ldots v_{-m}}_{v})_{(2)}$

wobei

- u_n : das Vorzeichenbit (0 für nichtnegative Zahl, 1 für negative Zahl)
- $u = u_{n-1} \ldots u_0$: eine ganze $\underline{n}$-Bit Zahl
- $v = v_{-1} \ldots v_{-m}$: eine rationale $\underline{m}$-Bit Zahl

Für diese Darstellung verwendet man die **Schreibweise**:

- $(+\underline{n}.\underline{m})$: $\underline{u_n}$ positives Vorzeichenbit, $\underline{n}$ Vorkommaziffern, $\underline{m}$ Nachkommaziffern
- $(-\underline{n}.\underline{m})$: $\underline{u_n}$ negatives Vorzeichenbit, $\underline{n}$ Vorkommaziffern, $\underline{m}$ Nachkommaziffern.

Beispiel:

$(0\,\underbrace{1010}_{u}\,.\,\underbrace{11}_{v})_{(2)}$ hat das Fixpunktdarstellungsformat $+4.2$.

Beispiele: Festpunktdarstellung

1. Umwandlung der Zahl $(0.125)_{(10)}$ in eine Binärzahl in $+\underline{2}.\underline{4}$-Festpunktdarstellung [d. h. positives Vorzeichenbit, $\underline{2}$ Vorkomma- $\underline{(n=2)}$ und $\underline{4}$ Nachkomma-Ziffern $\underline{(m=4)}$].

$$(0.125)_{(10)} = 0 + \frac{1}{8} = 0 + 2^{-3} = \underbrace{0}_{\text{Vorzeichen}} \; \underbrace{00}_{\text{2 Stellen}} \; . \; \underbrace{0010}_{\text{4 Stellen}}$$

Hier <u>sehr einfach</u>, da die Zahl durch eine einzige [negative] Zweierpotenz darstellbar ist. Die Umwandlung ist auch mittels sukzessiver Division für den Vorkommateil und sukzessiver Multiplikation für den Nachkommateil möglich (siehe folgende Beispiele).

Beispiele: Festpunktdarstellung

2. Umwandlung der Zahl $-1.3_{(10)}$ in eine Binärzahl in (-2.8)-Festpunktdarstellung.

   Zuerst die Umwandlung des Nachkommateils 0.3 (sukzessive Multiplikation):

	m	m' (mal 2)	b_k		
	0.3	$\underline{0}.6$	0	(weil < 1)	b_{-1}
	0.6	$\underline{1}.2$	1	(weil ≥ 1)	b_{-2}
	0.2	$\underline{0}.4$	0		b_{-3}
Der Prozess schliesst sich				Periode	
	0.4	$\underline{0}.8$	0		b_{-4}
	0.8	$\underline{1}.6$	1		b_{-5}
	0.6	$\underline{1}.2$	1		b_{-6}

Beispiele: Festpunktdarstellung (Fortsetzung Beispiel 2.)

- Die erste abgespaltete Ziffer b_k entspricht b_{-1}.
- Der Prozess terminiert nur dann, wenn die rationale Zahl als Summe negativer Zweierpotenzen darstellbar ist.
- Bei 0.3 ist dies nicht der Fall. Hier erhalten wir eine periodische Binärzahl.
  [Der Prozess schließt sich bei 0.6.]
  $\rightarrow$ daher die <u>Periode</u> 1001.
  Weil diese periodische Binärzahl auf $m = \underline{8}$ Stellen beschränkt ist:
  - $0.3 = 0.0\overline{1001} = 0.0\ \underbrace{1001}_{b_{-2}\dots b_{-5}}\ 100$
- Unter Beachtung des Vorzeichens ergibt sich
  - $-1.3_{(10)} = (\ \underbrace{1}_{\text{Vorzeichen}}\ \underbrace{01}_{\text{2 Stellen}}\ .\underbrace{01001100}_{\text{8 Stellen}}\ )_{(2)}$

Beispiele: Festpunktdarstellung

3. Umwandlung der Zahl $0.25_{(10)}$ in $+0.2$-Festpunktdarstellung.

   - $$\begin{array}{cccccc} m & m' & & & \\ 0.25 & \underline{0}.5 & 0 & b_{-1} & \big| \\ 0.5 & \underline{1}.0 & 1 & b_{-2} & \downarrow \end{array} \quad \to 0.25_{(10)} = (\underbrace{0}_{Vorzeichen} . \underbrace{01}_{2})_{(2)}$$

   Vorzeichen, 0 Vorkommastellen, 2 Nachkommastellen

Normierung

Die Zahl im Beispiel 2. können wir nicht exakt darstellen, auch wenn wir beliebig viele Stellen zur Verfügung hätten. Das führt zu Fehlern in numerischen Berechnungen in digitalen Rechenanlagen.

Eine Möglichkeit für die Verbesserung der Berechnungen mit Zahlen in Festpunktdarstellung ist die Normierung:

Die Nachkommastellen werden vor jedem Rechenschritt solange nach links geschoben, bis $b_{-1} \neq 0$ wird. Dies führt meist zu genauerer Berechnung, jedoch müssen alle Normierungen aufgezeichnet werden, um das Resultat korrekt denormieren zu können.

Zahlenumwandlung $B \rightarrow B^k$

B	:	Basis des Zahlensystems	(z. B. $B = 2, 7, \ldots$)
B^k	:	Basis eines „verwandten Zahlensystems“	(z. B. $2^3 = 8, 7^3 = 343$)

Umwandlung: Angenommen d ist eine rationale Zahl zur Basis B

$$\underset{\longleftarrow}{(d_{n-1} \ldots d_4 d_3 d_2 d_1 d_0} \quad \cdot \quad \underset{\longrightarrow}{d_{-1} d_{-2} d_{-3} d_{-4} \ldots d_{-m})_{(B)}}$$

Die Basis (neu): $B^{\underline{k}}$ $\quad \underline{k} = 2, 3, \ldots$

Zahlenumwandlung $B \to B^k$

- Für den Vorkommateil: Beginnend von $\underline{d_0}$ werden Teilketten von Bits der Länge $\underline{k}$ genommen und zur Basis B^k dargestellt $\to$ neue Ziffern für den Vorkommateil.
- Für den Nachkommateil: Beginnend mit $\underline{d_{-1}}$ werden analog k-Bit-Ketten zur Basis B^k dargestellt $\to$ neue Ziffern für den Nachkommateil.

Zahlenumwandlung $B \to B^k$

Begründung

- Vorkommateil transformiert [zur Basis B^k] durch sukzessive Division durch B^k [d. h. Schieben um k Stellen nach rechts].

  Ermittlung der ersten Ziffer:
  $(d_{n-1} \dots d_0)_B$: $B^k = d_{n-1} \dots d_k . d_{k-1} \dots d_0$
  $\to$ Rest der Division [d. h. die erste Ziffer z_0] zur Basis B^k ist $(d_{k-1} \dots d_0)_B = (z_0)_{B^k}$
  Also die erste k-Bit-Kette (Vorkomma, $\leftarrow \cdot$) dargestellt zur Basis B^k.

- Nachkommateil transformiert [zur Basis B^k] durch sukzessive Multiplikation mit B^k [d. h. Schieben um k Stellen nach links].

  Ermittlung der ersten Ziffer:
  $.d_{-1}d_{-2} \dots d_{-k}d_{-k-1} \dots d_{-m} \; \cdot \; B^k = d_{-1}d_{-2} \dots d_{-k}.d_{-k-1} \dots d_{-m}$
  $\to$ Die Zahl $(d_{-1} \dots d_{-k})_B$ ist die erste Nachkommaziffer $(z_{-1})_{(B^k)}$.

Zahlenumwandlung $B \to B^k$

Beispiel 1: $B = 2$

$(1000000, 1011)_2 \to \quad d_{n-1} = 1 \quad n = 8$
$\qquad d_{-m} = 1 \quad m = 4$

- oktale Darstellung (Umwandlung zur Basis $8 = 2^3 \to \underline{k = 3}$)

$$\begin{array}{ccccc} 1| & 000| & 000, & 101| & 1^{00} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1_8 & 0_8 & 0_8 & 5_8 & 4_8 \end{array} = (100, 54)_8$$

$$\begin{array}{c} 0_8 \\ \downarrow \\ (000)_2 = 0_8 \end{array}$$

- hexadezimale Darstellung (neue Basis $= 16 = 2^4 \to \underline{k = 4}$)

$$\begin{array}{ccc} 100| & 0000, & \underbrace{1011} \\ \downarrow & \downarrow & \downarrow \\ 4_{16} & 0_{16} & B_{16} \end{array} = (40, B)_{16} = 4 \cdot 16^1 + 0 + \tfrac{11}{16} = 64\tfrac{11}{16}_{(10)}$$

$$= (\underbrace{1000000}_{64}, \underbrace{1011}_{\frac{1}{2} + \frac{1}{8} + \frac{1}{16} = \frac{11}{16}})_2$$

- Umwandlung zur Basis $64 = 2^6 \to \underline{k = 6}$

$$\underbrace{1}_{1_{64}}|\underbrace{000000}_{0_{64}}, \underbrace{1011^{00}}_{44_{64}} = 10, 44_{(64)}$$

Zahlenumwandlung $B \to B^k$

Beispiel 2: $B = 7$

214_7
- Darstellung zur Basis $B^2 = 49\underline{(k = 2)}$

$$\begin{array}{cc} 2 | & 14_7 \\ \downarrow & \downarrow \\ 2_{49} & 11_{49} \end{array} \quad \Rightarrow 214_7 = 2\,11_{49} = 11 + 2 \cdot 49^1 = 109_{10} = 2 \cdot 7^2 + 1 \cdot 7^1 + 4 \cdot 7^0 = 214_7$$

Beispiel 3: $B = 3$

$22102,2110_3$
- Basis $9 = 3^2 \to \underline{k = 2}$

$$\begin{array}{ccccccccc} 2 & | & 21 & | & 02, & 21 & | & 10 \\ \downarrow & & \downarrow & & \downarrow & \downarrow & & \downarrow \\ 2_9 & & 7_9 & & 2_9 & 7_9 & & 3_9 \end{array} \quad \Rightarrow \; 272,73_{(9)}$$

Zahlenumwandlung $B1 \to B2$ (Allgemein)

$B1$, $B2$: Basen zweier unterschiedlicher Zahlensysteme.

Umwandlung $B1 \to B2$ mittels sukzessiver Division [Vorkommateil] und sukzessiver Multiplikation [Nachkommateil].

$\to$ Division (Multiplikation) mit $B2$ aber zur Basis $B1$ dargestellt, also mit $(B2)_{B1}$ [damit erfolgt die Operation im Zahlensystem zur Basis $B1$].

Die Ziffern der neuen transformierten Zahl zur Basis $B2$ sind:

- Rest bei Division (in $B2$-Darstellung) [Vorkommateil]
- Ganzzahliger Teil des Produkts ($B2$-Format) [Nachkommateil]

Beispiel 1: $B1 = 3 \quad B2 = 7 \quad 110_3 \stackrel{?}{=} x_7$

$$
\begin{array}{ll}
110_3 : 7 = 110_3 : 21 = 1 & 12_3 = 5_7 \\
\quad -21_3 & \\
\quad \overline{12_3} = Rest & \\
1_3 : 21_3 = 0 & \\
1_3 = Rest & 1_3 = 1_7
\end{array}
$$

$$\Rightarrow x_7 = 15_7 = 12_{10} = 110_3$$

Beispiel 2: $B1 = 5 \quad B2 = 4 \quad 4013_5 \stackrel{?}{=} x_4$

$$
\begin{array}{ll}
4013_5 : 4 = 4013_5 : 4_5 = 1002_5 & \\
-4_5 & \\
\overline{0013_5} & \\
-13_5 & \\
\overline{0_5} = Rest & 0_5 = 0_4 \\[1em]
1002_5 : 4_5 = 111_5 & \\
-4_5 & \\
\overline{10_5} & \\
-4_5 & \\
\overline{12_5} & \\
-4_5 & \\
\overline{3_5} = Rest & 3_5 = 3_4 \\[1em]
111_5 : 4_5 = 12_5 & \\
-4_5 & \\
\overline{21_5} & \\
-13_5 & \\
\overline{3_5} = Rest & 3_5 = 3_4 \\[1em]
12_5 : 4_5 = 1_5 & \\
-4_5 & \\
\overline{3_5} = Rest & 3_5 = 3_4 \\[1em]
1_5 : 4_5 = 0 & \\
1_5 = Rest & 1_5 = 1_4
\end{array}
$$

$\Rightarrow$

$$x_4 = 13330_4 =$$

$$= 0 + 12 + 3 \cdot 16 + 3 \cdot 64 + 256 = 508_{(10)}$$

$$4013_5 = 3 + 5 + 4 \cdot 125 = 508_{(10)}$$

Beispiel 3: $B1 = 5 \quad B2 = 10 \quad 0{,}4_5 \stackrel{?}{=} x_{10}$

$$\begin{array}{l} 0,4_5 \cdot 10 = 0,4_5 \cdot 20_5 = \\ \qquad\qquad\quad 13_5 \\ \qquad\qquad\quad\ \ \underline{\ \ 0_5} \\ \qquad\qquad\quad \underline{13},0_5 \Rightarrow \underline{8}_{(10)} \quad \text{ist die erste Nachkomma Ziffer zur Basis 10} \end{array}$$

$$\Rightarrow \quad 0,4_5 = 0,8_{10}$$

Probe: $0{,}4_5 = \frac{4}{5} = \underline{0{,}8_{10}} = x_{10}$

Beispiel 4: $B1 = 4 \quad B2 = 6 \quad 0{,}312_4 \stackrel{?}{=} x_6$

$$0,312_4 \cdot 6 = 0,312 \cdot 12_4$$
$$\frac{1230_4}{11,010_4} \quad \Rightarrow \quad 11_4 = 5_6$$
$$0,010_4 \cdot 12_4 = \frac{020_4}{0,120_4} \quad \Rightarrow \quad 0_4 = 0_6$$
$$0,120_4 \cdot 12_4 = \frac{300_4}{0,2100_4} \quad \Rightarrow \quad 0_4 = 0_6$$
$$0,21_4 \cdot 12_4 = \frac{102_4}{0,312_4} \quad \Rightarrow \quad 0_4 = 0_6$$

$\Rightarrow x_6 = 0.\overline{5000}_6$

$\Rightarrow$ 5000 ist die Periode

$\ldots. \Rightarrow$ Periode

Probe: $0{,}312_4 = \frac{3}{4} + \frac{1}{16} + \frac{2}{64} \approx \frac{13}{16}$ $\qquad 0,\overline{5000}_6 \approx \frac{5}{6}$

$\frac{13}{16} - \frac{5}{6} = \frac{78-80}{96} = \frac{-2}{96} = \frac{-1}{48} \rightarrow$ ok.

Gleitpunktdarstellung rationaler Zahlen

Effiziente Darstellung rationaler Zahlen durch Gleitpunktdarstellung (Floating-Point-Representation)

Grundidee:

- $q = \pm M \cdot B^C$
- B : Basis [z. B.: $B = 2$, $B = 10$]
- M : Mantisse
- C : Charakteristik

Beispiel: (Fix-Point versus Floating-Point-Darstellung)

- weniger Zeichen, $q = 2^{-16}$
  Wahl $B = 2$, $M = 1.0$

Festpunkt: $q_{(2)} = (0.00\ldots\underbrace{1}_{b_{-16}})_{(2)} \rightarrow$ 16 Nachkommastellen

Floating-Point: Für $B = 2$ und $M = 1.0$ benötigt man nur 6 Bit:

5 Bit für C $[16_{(10)} = (10000)_2]$
1 Bit für M $[1.0_{(10)} = (1)_2]$
$+$ 1 Bit [Vorzeichenbit für C].

Eine andere Wahl von M [z. B. $M = 2.0$] würde zu einer anderen Darstellung führen.

Floating Point Standards

IEEE-Standards [„EI-TRIPL-I"] - Institute of Electrical and Electronic Engineers [weltweit renommierter Fachverein]. Diese Standards gelten für die Repräsentation von Zahlen in modernen Rechnern.

Format:

- 32 Bit Wortlänge: single precision
- $2 \cdot$ 32-Bit-Wörter [= 64 Bit]: double precision
- $4 \cdot$ 32-Bit-Wörter [oder $2 \cdot$ 64-Bit-Wörter = 128 Bit]: quadruple precision

Wir zeigen das 32- und das 64-Bit-Format.

IEEE-32-Bit-Format

v	C	m
1	8	23

v : das Vorzeichenbit [$v = 0$ oder $v = 1$]

C : die <u>8</u>-stellige Charakteristik

$$C = (c_7c_6...c_0)_{(2)} = c_7 \cdot 2^7 + c_6 \cdot 2^6 + ... + c_0 \cdot 2^0, \quad c_i \in \{0, 1\}$$

$$\rightarrow C = 0, 1, ..., 2^8 - 1 = 255$$

[also eine übliche binäre Darstellung]

IEEE-32-Bit-Format

v	C	m
1	8	23

m : <u>23</u>-stelliger Binärbruch $m_{22}m_{21}...m_0$ der Mantisse
$M = (1.m)_{(2)}, m_i \in \{0,1\}$ [mit Komma in der Darstellung ganz links]

d. h. $M = (\underline{1}.m)_{(2)} = (\underline{1}.r)_{(10)} = (\underline{1} + r)_{(10)}, 0 \leq \underline{r} < 1$

weil

$$M = (\underline{1}.m)_{(2)} = (\underline{1} + \underbrace{m_{22} \cdot 2^{-1} + m_{21} \cdot 2^{-2} + ... + m_0 \cdot 2^{-23}}_{r})_{(10)}$$

$$= (\underline{1} + r)_{(10)} = (1.r)_{(10)}$$

also $\underline{r_{(10)} = (m)_{(2)}}$

IEEE-32-Bit-Format

Die Darstellung [Repräsentation] einer rationalen Zahl q im 32-Bit-IEEE-Format:

$$q_{\text{IEEE-32-Bit}} = (\underbrace{v}_{1}\ \underbrace{c_7 \ldots c_0}_{8}\ \underbrace{m_{22}\, m_{21} \ldots m_0}_{23})_{\text{IEEE-32-Bit}}$$

Die Zahl q, repräsentiert durch die obige Darstellung, wird dezimal interpretiert als

$$q_{(10)} = (-1)^v \cdot 2^{(C-127)_{(10)}} \cdot (1.m)_{(2)} = (-1)^v \cdot 2^{C'_{(10)}} \cdot (1.r)_{(10)},$$

wobei $C' = (C - 127)_{(10)}$ die Interpretation von $C \in [0 \ldots 255]$ in Excess-127-Code ist.

IEEE-32-Bit-Format

Also:

Die Charakteristik C [repräsentiert in $q_{\text{IEEE-32-Bit}}$ durch 8 Bits $c_7 \ldots c_0$] wird in der Dezimaldarstellung $q_{(10)}$ von q interpretiert als $C' = (C - 127)_{(10)}$.

Konkret:

C	C'
0	-127
1	-126
...	...
127	0
...	...
255	128

Ausnahmefälle

Wenn $C = (\overbrace{00000000}^{8}) = 0_{(10)}$, d. h. $C' = (-127)_{(10)}$, wird M ohne 1 in der Mantisse [also $M = (\underline{0}.m)_{(2)} = (0.r)_{(10)}$] interpretiert.

1. Speziell [Darstellung von $0_{(10)}$]:
   Wenn $C = 00000000$ und $m = \underbrace{00000000000000000000000}_{23}$, dann ist die dargestellte Zahl $q_{(10)} = \underline{0}$
   [0 hat zwei Darstellungen: $v = 0 : +0$, $v = 1 : -0$]

2. $C = 00000000\ (= 0), m \neq 0$ [„denormalized numbers to fill in gap around zero"]
   $\rightarrow q_{(10)} = (-1)^v \cdot 2^{\underline{-126}} \cdot (0.m)_{(2)} = (-1)^v \cdot 2^{\underline{-126}} \cdot (0.r)_{(10)}$

Ausnahmefälle

3. $C = 11111111\ (= 255),\ m = \underbrace{00000000000000000000000}_{23}\ (= 0)$
   $\rightarrow q_{(10)} = (-1)^v \cdot \infty\ [v = 0 : +\infty, v = 1 : -\infty]$

4. $C = 11111111\ (= 255),\ m \neq 0$
   $\rightarrow q_{(10)} = NaN$ [Not a Number], symbolisiert ungültige Ergebnisse
   [z. B. Division durch 0].

Beispiele: Umwandlung von $q_{\text{IEEE-32-Bit}}$ in die Dezimalform

- **Interpretation** der 32-Bit-Binärzahl

$$q_{\text{IEEE-32-Bit}} = \overbrace{1}^{v}\ \overbrace{10000111}^{C}\ \overbrace{10100000000000000000000}^{m} \text{ im Dezimalsystem.}$$

  - $v = 1$ [$\rightarrow$ negatives Vorzeichen]
  - $C = (10000111)_{(2)} = (135)_{(10)}$
  - $M = (1.m)_{(2)} = (1.\underbrace{10100\ldots0}_{m})_{(2)} = 1 + \underbrace{2^{-1} + 0 + 2^{-3} + 0\ldots}_{\underline{r}} = (1.\underline{625})_{(10)}$

  $\rightarrow q_{(10)} = (-1)^1 \cdot 2^{135-127} \cdot 1.625 = -2^8 \cdot 1.625 = (-416)_{(10)}$

  - Wäre $C = (01111000)_{(2)}$ [$= 120_{(10)}$], dann ist $C' = C - 127 = (-7)_{(10)}$ und die Floating-Point-Darstellung wäre $-2^{-7} \cdot 1.625 \approx (-.0127)_{(10)}$.
  - Falls $C = (00000000)_{(2)}$ wäre $\rightarrow$ [laut <u>Ausnahme</u>]
  $q_{(10)} = (-1)^1 2^{\underline{-126}} \cdot \underline{0}.\underline{625} \approx -7.34 \cdot 10^{-39}$
  [<u>also:</u> 1 [Vorkommastelle] ist in der Mantisse weggelassen]

Beispiele: Umwandlung von $q_{\text{IEEE-32-Bit}}$ in die Dezimalform

- Interpretation der Zahl $(0\ 01111000\ \underline{1}1111000000000000000000)_{\text{IEEE-32-Bit}}$ im Dezimalsystem.
  - $v = (0)_{(2)}$
  - $C = (01111000)_{(2)} = 2^6 + 2^5 + 2^4 + 2^3 = (120)_{(10)}$

    $\rightarrow C' = C - 127 = 120 - 127 = (-7)_{(10)}$
  - $r_{(10)} = (.m)_{(2)} = \underline{1} \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} = (0.96875)_{(10)}$

  $\rightarrow q_{(10)} = (-1)^0 \cdot 2^{-7} \cdot 1.96875 \approx 0.015$

Umwandlung von Dezimalzahlen ins IEEE-Format

1. $q_{(10)}$ in die gewünschte Form bringen:

   $\boxed{\cdot}$ $q_{(10)} = (-1)^{v_{(2)}} \cdot 2^{C'_{(10)}} \cdot (1.m)_{(2)}$

   Dazu wandeln wir $q_{(10)}$ zunächst in die Binärform um:

   $q_{(10)} = q_{(2)} = (\pm b_\ell b_{\ell-1} \ldots b_0 . b_{-1} b_{-2} \ldots b_k \ldots)_{(2)}$

   [durch sukzessive Division (Vorkommateil) von $q_{(10)}$ und sukzessive Multiplikation (Nachkommateil)], wobei $\underline{b_\ell = 1}$

   $\rightarrow q_{(10)} = (\pm 1) \cdot 2^{(\ell)_{(10)}} \cdot (1.b_{\ell-1} \ldots b_0 b_{-1} b_{-2} \ldots b_k \ldots)_{(2)}$

   Aus $\boxed{\cdot}$ folgt:

   - $v = 0$ $(+)$ oder $v = 1$ $(-)$
   - $C' = (\ell)_{(10)}$
   - $(1.m)_{(2)} = (1.b_{\ell-1} b_{\ell-2} \ldots b_k \ldots)_{(2)}$

Umwandlung von Dezimalzahlen ins IEEE-Format

2. aus der gewünschten Form $\boxdot$ das entsprechende IEEE-32- bzw. 64-Bit-Format generieren:
   z. B. 32 Bit $q_{\text{IEEE-32-Bit}} = (v \ C \ m)_{\text{IEEE-32-Bit}}$
   - $v = v_{(2)}$ (1 Bit)
   - $C = (C' + 127)_{(10)} = (\ell + 127)_{(2)}$ (8 Bit)
   - $m = m_{(2)}$ (23 Bit)

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(29.0)_{(10)}$ im IEEE-32-Bit-Format

1. Umwandlung von $q_{(10)} = (29.0)_{(10)}$ in die gewünschte Form
   $\boxdot$ $q_{(10)} = (-1)^{v_{(2)}} \cdot 2^{C'_{(10)}} \cdot (1.m)_{(2)}$

   Zunächst die Darstellung von $q_{(10)}$ in Binärform:
   $$\begin{array}{ccccc|c} 29 & : & 2 & = & 14 & 1 \\ 14 & : & 2 & = & 7 & 0 \\ 7 & : & 2 & = & 3 & 1 \\ 3 & : & 2 & = & 1 & 1 \\ 1 & : & 2 & = & 0 & 1 \end{array}$$

   $(29.0)_{(10)} = (11101.0)_{(2)} = (+b_4 b_3 b_2 b_1 b_0)_{(2)} \to \ell = (4)_{(10)}$
   $\boxdot$ $(29.0)_{(10)} = (-1)^0 \cdot 2^4 \cdot (1.1101)_{(2)}$
   $\to v = 0, \quad C' = 4_{(10)}, \quad (1.m)_{(2)} = (1.1101)_{(2)}$.

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(29.0)_{(10)}$ im IEEE-32-Bit-Format (Fortsetzung)

2. Aus der gewünschten Form $\boxdot$ das IEEE-32-Bit-Format für $(29.0)_{(10)}$ ableiten.

   - $v = 0_{(2)}$ (1 Bit)
   - $C = (C' + 127)_{(10)} = (4 + 127)_{(10)} = (131)_{(10)} = (10000011)_{(2)}$ (8 Bit)
   - $m = (11010\ldots0)_{(2)}$ (23 Bit)

$$\rightarrow q_{\text{IEEE-32-Bit}} = (v\ \ C\ \ m)_{\text{IEEE-32-Bit}}$$
$$= (0\ 10000011\ 11010000000000000000000)_{\text{IEEE-32-Bit}}$$

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(0.5)_{(10)}$ im IEEE-32-Bit-Format

1. Gewünschte Form
   Die binäre Form:
   $(0.5)_{(10)} = (0.1)_{(2)}$ $\qquad 0.5 \cdot 2 = 1.0 \qquad 1 = b_{-1}$
   $\rightarrow \ell = (-1)_{(10)}$.
   $\rightarrow$ Die gewünschte Form: $(0.5)_{(10)} = (-1)^0 \cdot 2^{-1} \cdot (1.0)_{(2)}$
   $\rightarrow v = 0, \quad C' = (-1)_{(10)}, \quad (1.m)_{(2)} = (1.0)_{(2)}$
2.
   - $v = 0$
   - $C = (C' + 127)_{(10)} = (-1 + 127)_{(10)} = (126)_{(10)} = (01111110)_{(2)}$
   - $m = (0 \ldots 0)_{(2)}$ (23 Bit)

$(0.5)_{(10)} = (0\ 01111110\ 00000000000000000000000)_{\text{IEEE-32-Bit}}$

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(-1.0)_{(10)}$ im IEEE-SP-Format

1. Gewünschte Form
   $(-1.0)_{(10)} = (-1)^1 \cdot 2^0 \cdot (1.0)_{(2)} \rightarrow$
   $v = 1, \quad \ell = C' = 0, \quad .m = 0 \ldots 0$
2. $v = 1 \quad C = C' + 127 = 127 = 011 \ldots 1 \quad m = \underbrace{0 \ldots 0}_{23}$

   $(-1.0)_{(10)} = 1\ 01111111\ 00000000000000000000000 \quad \circledast$

Bemerkung:
Darstellung von $(1.0)_{(10)}$ wie $\circledast$ nur v=0

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(-\underline{47}.\underline{7})_{(10)}$ im IEEE-32-Bit-Format

$$
\begin{array}{lccccc|ccc}
 & \mathrm{m} & & & & m' & b_i & & \\
 & \underline{47} & : & 2 & = & 23 & 1 & b_0 & \uparrow \\
 & 23 & : & 2 & = & 11 & 1 & b_1 & | \\
 & 11 & : & 2 & = & 5 & 1 & b_2 & | \\
 & 5 & : & 2 & = & 2 & 1 & b_3 & | \\
 & 2 & : & 2 & = & 1 & 0 & b_4 & | \\
 & 1 & : & 2 & = & 0 & 1 & b_5 & | \\
 & & & & & & & & \\
 & \underline{0.7} & \cdot & 2 & = & 1.4 & 1 & b_{-1} & | \\
\lceil & 0.4 & \cdot & 2 & = & 0.8 & 0 & b_{-2} & | \\
\text{Periode} \; | & 0.8 & \cdot & 2 & = & 1.6 & 1 & b_{-3} & | \\
| & 0.6 & \cdot & 2 & = & 1.2 & 1 & b_{-4} & | \\
\lfloor & 0.2 & \cdot & 2 & = & 0.4 & 0 & b_{-5} & | \\
 & 0.4 & \cdot & 2 & = & 0.8 & 0 & b_{-6} & | \\
 & 0.8 & \cdot & 2 & = & 1.6 & 1 & b_{-7} & | \\
 & 0.6 & \cdot & 2 & = & 1.2 & 1 & b_{-8} & \downarrow \\
 & & & & & \vdots & \vdots & &
\end{array}
$$

Umwandlung von Dezimalzahlen ins IEEE-Format

Beispiel: Darstellung von $(-\underline{47}.\underline{7})_{(10)}$ im IEEE-32-Bit-Format (Fortsetzung)

1. Gewünschte Form:

$$(-47.7)_{(10)} = (\underbrace{b_5 b_4 b_3 b_2 b_1 b_0}_{101111} . \underbrace{b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} \ldots}_{1\underline{0110}\ldots})_{(2)}$$

$$= (-1) \cdot 2^5 \cdot (1.b_4 b_3 b_2 b_1 b_0 b_{-1} b_{-2} b_{-3} \ldots)_{(2)}$$

$$= (-1) \cdot 2^5 \cdot (1.01111 \underbrace{\ 1\ }_{b_{-1}} \underline{0110}0 \ldots)_{(2)}$$

2.
   - $v = (1)_{(2)}$ (1 Bit)
   - $C' = \ell = (5)_{(10)} \rightarrow C = 127 + 5 = (132)_{(10)} = (10000100)_{(2)}$ (8 Bit)
   - $m = 011111011001 \ldots = b_{-1} b_{-2} \ldots b_k \ldots = b_4 b_3 b_2 \ldots$ (23 Bit)

$$(-47.7)_{(10)} = (1\ 10000100\ 011111\,\underline{0110}\,\underline{0110}\,\underline{0110}\,\underline{0110}\,0)_{\text{IEEE-32-Bit}}$$

Umwandlung von Dezimalzahlen ins IEEE-Format

[Die größte positive Single-Precision-Zahl im IEEE-Format: 32 Bit]

Positive Zahl:

- $v = 0$
- $C = (1111111\underline{0})_{(2)} = \overline{(254)_{(10)}}$
  [weil $C = 11111111$ und $\overline{m \neq 0}$ für den Ausnahmefall 4. reserviert ist]
- $M = (1.m)_{(2)} = (1.\underbrace{111\ldots1}_{23\times})_{(2)}$

$$
\begin{array}{lcl}
q_{\text{IEEE-32-Bit}} & = & (0\ 11111110\ 11111111111111111111111) \\
\text{[largest positive]} & = & (-1)^0 \cdot 2^{(C-127)_{(10)}} \cdot (1.m)_{(2)} \\
 & = & 2^{(254-127)_{(10)}} \cdot (1.\underbrace{111\ldots1}_{23\times})_{(2)} \\
 & = & 2^{127} \cdot (2 - 2^{-23})_{(10)} \approx (3.4028 \cdot 10^{38})_{(10)}
\end{array}
$$

Umwandlung von Dezimalzahlen ins IEEE-Format

[Die größte positive Single-Precision-Zahl im IEEE-Format: 32 Bit (Fortsetzung)]

Beweis.

für $(1.\underbrace{111\ldots1}_{23\times})_{(2)} = (2-2^{-23})_{(10)}$

$$\begin{aligned}(1.\underbrace{111\ldots1}_{23\times})_{(2)} &= 1+2^{-1}+2^{-2}+2^{-3}+\ldots+2^{-23} = \\ &= 2^0+2^{-1}+2^{-2}+2^{-3}+\ldots+2^{-23} = \sum_{i=0}^{23}\left(\tfrac{1}{2}\right)^i\end{aligned}$$

Es gilt für die Summe der geometrischen Reihe: $\sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \rightarrow$

$$\sum_{i=0}^{23}\left(\frac{1}{2}\right)^i = \frac{1-(\frac{1}{2})^{24}}{1-\frac{1}{2}} = \frac{1-(\frac{1}{2})^{24}}{\frac{1}{2}} = 2(1-2^{-24}) = (2-2^{-23})_{(10)}\,.$$

Die 32-Bit-Darstellung erlaubt nur eine eingeschränkte Genauigkeit $\rightarrow$ daher definiert man auch ein 64-Bit-IEEE-Format (Double Precision). □

IEEE-64-Bit-Format

Die binäre Darstellung [Repräsentation]:
$q_{\text{IEEE-64-Bit}} = \underbrace{v}_{1}\,\underbrace{c_{10}\ldots c_0}_{11}\,\underbrace{m_{51}\ldots m_0}_{52}$ wird als Dezimalzahl

$$q_{(10)} = (-1)^v \cdot 2^{C-1023} \cdot (1.m)_{(2)} = (-1)^v \cdot 2^{C'} \cdot (1.r)_{(10)}$$

interpretiert.

Bemerkung: Im IEEE-Format ist die Bezeichnung „Floating-Point" etwas irreführend, da in dieser Darstellung das Komma gar nicht explizit auftritt.

IEEE-64-Bit-Format

Darstellung von $(-3.5)_{(10)}$ im IEEE-64-Bit-Format

$$\begin{aligned}(-3.5)_{(10)} &= (11.1)_{(2)} \\ &= (b_{\underline{1}} b_0 . b_{-1})_{(2)}\end{aligned} \quad \left| \quad \begin{array}{llll} 3:2 & = & 1 & 1 \\ 1:2 & = & 0 & 1 \\ 0.5 \cdot 2 & = & 1.0 & 1 \end{array} \right.$$

$$\rightarrow \ell = \underline{1}$$

$$\rightarrow (-3.5)_{(10)} = (-1)^1 \cdot 2^{\underline{1}} \cdot (1.11)_{(2)}$$

$$\rightarrow v = 1 \quad C' = \ell = 1 \quad (1.m)_{(2)} = (1.11)_{(2)}$$

$$\rightarrow \begin{array}{rcl} C & = & (C' + 1023)_{(10)} = (1 + 1023)_{(10)} = (1024)_{(10)} = 2^{10} \\ & = & (10000000000)_{(2)} \ (11 \text{ Bit}) \end{array}$$

$$\rightarrow m = (110\ldots0)_{(2)} \ (52 \text{ Bit})$$

$$(-3.5)_{(10)} = (1 \ 1\underbrace{000\ldots0}_{10\times} \ 11\underbrace{000\ldots0}_{50\times})_{\text{IEEE-64-Bit}}$$