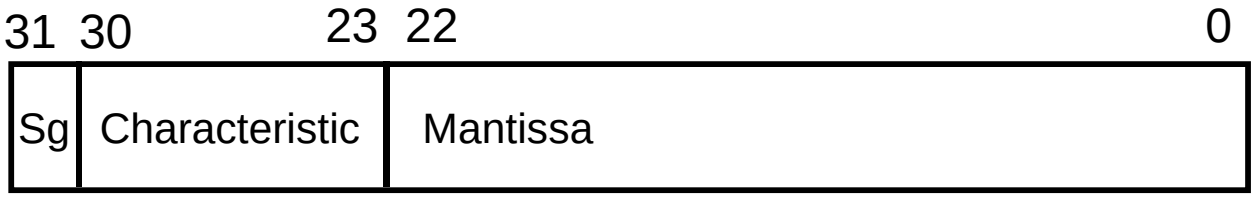
Prof. Dr.-Ing. Jochen Schiller
Computer Systems & Telematics



TI II: Computer Architecture
Data Representation and Computer Arithmetic

- Systems
- Representations
- Basic Arithmetic
- ALU

$$(x + y) + z \neq x + (y + z)$$

Rechnerarithmetik

Grundlagen Technische Informatik: „Verarbeitung“ einzelner Bits (Schaltnetze/Schaltwerke)

Rechnerarithmetik soll nun als Beispiel dienen, wie größere Informationseinheiten verarbeitet werden

Hierzu werden zunächst die formalen Grundlagen erarbeitet

Danach werden Verfahren und Schaltungen zur Implementierung der 4 Grundrechenarten in einem Rechner vorgestellt

Abschließend wird die Funktion einer arithmetisch logischen Einheit (ALU) eines Rechners besprochen

Formale Grundlagen

Menschen: rechnen gewöhnlich im Dezimalzahlensystem

Rechner: rechnen gewöhnlich im Dualzahlensystem

➡ eine Konvertierung ist erforderlich

Daneben werden weitere Zahlensysteme wie Oktalzahlensystem oder Hexadezimalzahlensystem (eigentlich: Sedezimal) zur kompakteren Darstellung der sehr langen Dualzahlen verwendet.

- Es ist notwendig, die Zusammenhänge und mathematischen Grundlagen dieser Zahlensysteme zu verstehen

Siehe Mathematik für Informatiker!

- Hier nur im Überblick bzw. Schnelldurchgang

ZAHLENSYSTEME

Anforderungen an Zahlensysteme

Positive und negative Zahlen eines Intervalls [-x : y] sollen dargestellt werden können

Es sollen etwa gleich viele positive wie negative Zahlen darstellbar sein

Die Darstellung sollte eindeutig sein

Die Darstellung sollte die Durchführung von Rechenoperationen erlauben bzw. vereinfachen

- Römische Zahlen
- Chinesische Zahlen
- Arabische Zahlen

Zahlensysteme

Gängigste Form: Stellenwertsysteme

Zahlendarstellung in Form einer Reihe von Ziffern z_i, wobei das Dezimalkomma rechts von z_0 platziert sei:

- $z_n\ z_{n\text{-}1}\ \ldots\ z_1\ z_0\ ,\ z_{\text{-}1}\ z_{\text{-}2}\ \ldots\ z_{\text{-}m}$ z.B. 1234,567

Jeder Position i der Ziffernreihe ist ein Stellenwert zugeordnet, der eine Potenz b^i der Basis b des Zahlensystems ist.

- b-näres Zahlensystem

Der Wert X_b der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen $z_i b^i$:

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \cdots + z_1 b^1 + z_0 b^0 + z_{-1} b^{-1} + \cdots + z_{-m} b^{-m} = \sum_{i=-m}^{n} z_i b^i$$

Zahlensysteme

Interessante Zahlensysteme in der Informatik

Basis (b)	Zahlensystem	Zahlenbezeichnung	Alphabet
2	Dualsystem	Dualzahl	0,1
8	Oktalsystem	Oktalzahl	0,1,2,3,4,5,6,7
10	Dezimalsystem	Dezimalzahl	0,1,2,3,4,5,6,7,8,9
16	Hexadezimalsystem (Sedezimalsystem)	Hexadezimalzahl (Sedezimalzahl)	0,1,2,3,4,5,6,7,8,9,A,B, C,D,E,F

Hexadezimalsystem: Die „Ziffern“ 10 bis 15 werden mit den Buchstaben A bis F dargestellt.

Dualsystem: Wichtigstes Zahlensystem im Rechner

Oktal- und Hexadezimalsystem: Leicht ins Dualsystem umwandelbar, besser zu verstehen als lange 0-1-Kolonnen.

UMWANDLUNG IN B-NÄRE ZAHLENSYSTEME

Der Euklidische Algorithmus

Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis b

1. Methode: Euklidischer Algorithmus:

$$Z = z_n\,10^n + z_{n-1}\,10^{n-1} + \ldots + z_1\,10 + z_0 + z_{-1}\,10^{-1} + \ldots + z_{-m}\,10^{-m}$$

$$= y_p\,b^p + y_{p-1}\,b^{p-1} + \ldots + y_1\,b + y_0 + y_{-1}\,b^{-1} + \ldots + y_{-q}\,b^{-q}$$

Die Ziffern werden sukzessive, beginnend mit der höchstwertigen Ziffer, berechnet:

1. Schritt: Berechne $\boldsymbol{p}$ gemäß der Ungleichung $b^p \le Z < b^{p+1}$

   setze $i = p$ und $Z_i = Z$

2. Schritt: Ermittle $\boldsymbol{y_i}$ und den Rest $\boldsymbol{R_i}$ durch Division von $\boldsymbol{Z_i}$ durch $\boldsymbol{b^i}$

   $\boldsymbol{y_i} = \boldsymbol{Z_i}$ **div** $\boldsymbol{b^i}$

   $\boldsymbol{R_i} = \boldsymbol{Z_i}$ **mod** $\boldsymbol{b^i}$

3. Schritt: Wiederhole 2. Schritt für $i = p\text{-}1,\ldots$ und ersetze dabei nach jedem Schritt $\boldsymbol{Z_i}$ durch $\boldsymbol{R_i}$, bis $\boldsymbol{R_i} = \boldsymbol{0}$ oder bis $\boldsymbol{b^i}$ gering genug ist (und damit auch der Umrechnungsfehler).

Der Euklidische Algorithmus: Beispiel

Umwandlung von $15741{,}233_{10}$ ins Hexadezimalsystem

1. Schritt: $16^3 \leq 15741{,}233 < 16^4$ ➡ höchste Potenz 16^3

2. Schritt:	15741,233	:	16^3	=	3	Rest	3453,233
3. Schritt:	3453,233	:	16^2	=	D	Rest	125,233
4. Schritt:	125,233	:	16^1	=	7	Rest	13,233
5. Schritt:	13,233	:	$16^0=1$	=	D	Rest	0,233
6. Schritt:	0,233	:	16^{-1}	=	3	Rest	0,0455
7. Schritt:	0,0455	:	16^{-2}	=	B	Rest	0,00253
8. Schritt:	0,00253	:	16^{-3}	=	A	Rest	0,000088593
9. Schritt:	0,000088593	:	16^{-4}	=	5	Rest	0,000012299

(➡ Fehler)

➡ $\mathbf{15741{,}233_{10} \approx 3D7D{,}3BA5_{16}}$

Horner Schema

Umwandlung vom Dezimalsystem in ein Zahlensystem zur Basis b

2. Methode: Abwandlung des Horner Schemas

- Hierbei müssen der ganzzahlige und der gebrochene Anteil getrennt betrachtet werden.
- Umwandlung des ganzzahligen Anteils:
- Eine ganze Zahl $X_b = \sum_{i=0}^{n} z_i b^i$ kann durch fortgesetztes Ausklammern auch in folgender Form geschrieben werden:

$$X_b = ((...(((z_n\, b + z_{n\text{-}1})\, b + z_{n\text{-}2})\, b + z_{n\text{-}3})\, b \ldots)\, b + z_1)\, b + z_0$$

Horner Schema: Beispiel

Die gegebene Dezimalzahl wird sukzessive durch die Basis b **dividiert**.

Die jeweiligen ganzzahligen Reste ergeben die Ziffern der Zahl X_b in der Reihenfolge von der niedrigstwertigen zur höchstwertigen Stelle.

Wandle 15741_{10} ins Hexadezimalsystem

$15741_{10} : 16 = 983$ Rest 13 (D_{16})

$983_{10} : 16 = 61$ Rest 7 (7_{16})

$61_{10} : 16 = 3$ Rest 13 (D_{16})

$3_{10} : 16 = 0$ Rest 3 (3_{16})

➡ **$15741_{10} = 3D7D_{16}$**

Horner Schema: Umwandlung des Nachkommateils

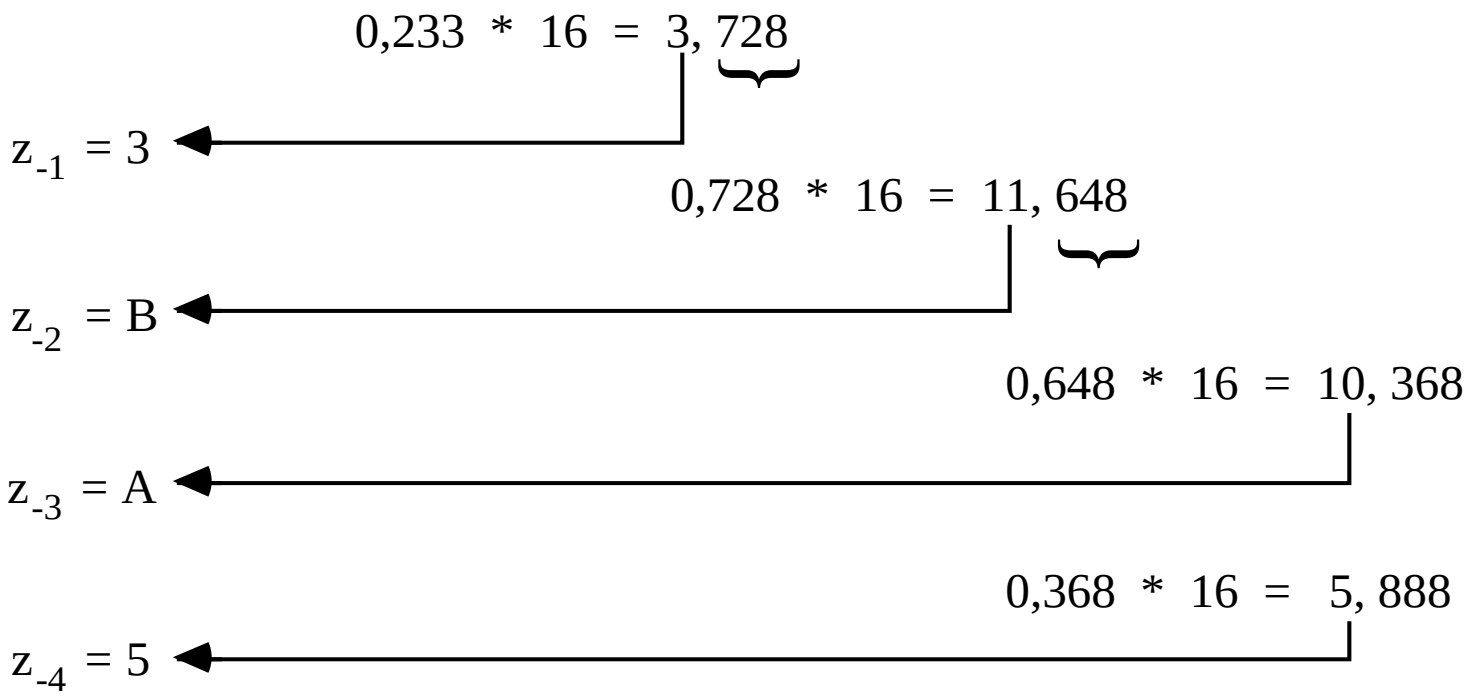
Auch der gebrochene Anteil einer Zahl $Y_b = \sum_{i=-m}^{-1} y_i b^i$ lässt sich entsprechend schreiben:

$$Y_b = ((...((y_{-m}\, b^{-1} + y_{-m+1})\, b^{-1} + y_{-m+2})\, b^{-1} + ... + y_{-2})\, b^{-1} + y_{-1})\, b^{-1}$$

Verfahren**:**
Eine sukzessive **Multiplikation** des Nachkommateils der Dezimalzahl mit der Basis b des Zielsystems ergibt nacheinander die y_{-i} in der Reihenfolge der höchstwertigen zur niedrigstwertigen Nachkommaziffer.

Horner Schema: Umwandlung des Nachkommateils – Beispiel

Umwandlung von $0{,}233_{10}$ ins Hexadezimalsystem:



Abbruch bei genügend hoher Genauigkeit

➡ **$0{,}233_{10} \approx 0{,}3BA5_{16}$**

UMWANDLUNG INS DEZIMALSYSTEM

Umwandlung: Basis $b \rightarrow$ Dezimalsystem

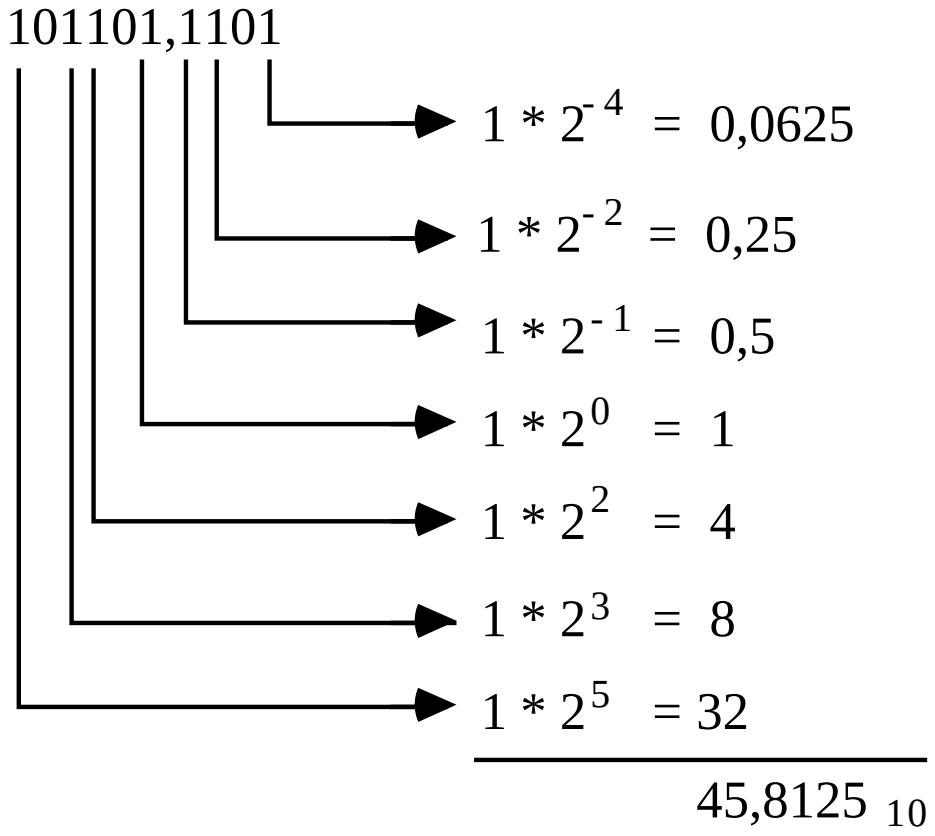
Die Werte der einzelnen Stellen der umzuwandelnden Zahl werden in dem Zahlensystem, in das umgewandelt werden soll, dargestellt und nach der Stellenwertgleichung aufsummiert.

Der Wert X_b der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen $z_i b^i$:

$$X_b = z_n b^n + z_{n-1} b^{n-1} + \cdots + z_1 b^1 + z_0 b^0 + z_{-1} b^{-1} + \cdots + z_{-m} b^{-m} = \sum_{i=-m}^{n} z_i b^i$$

Umwandlung: Basis $b \rightarrow$ Dezimalsystem – Beispiel

Konvertiere $101101{,}1101_2$ ins Dezimalsystem



Umwandlung beliebiger Stellenwertsysteme

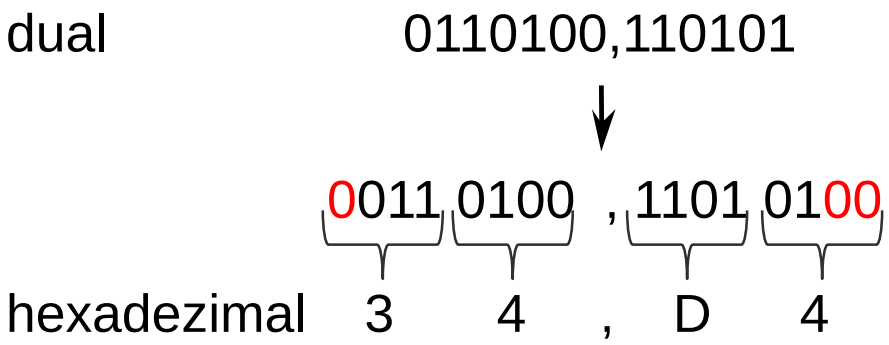
Man wandelt die Zahl ins Dezimalsystem um und führt danach mit Methode 1 oder 2 die Wandlung ins Zielsystem durch.

Spezialfall:

- Ist eine Basis eine Potenz der anderen Basis, können einfach mehrere Stellen zu einer Ziffer zusammengefasst werden oder eine Stelle kann durch eine Folge von Ziffern ersetzt werden.

Wandlung von $0110100{,}110101_2$ ins Hexadezimalsystem

$2^4 = 16 \Rightarrow$ 4 Dualstellen $\rightarrow$ 1 Hexadezimalstelle



Ergänzen von Nullen zum Auffüllen auf Vierergruppen

NEGATIVE ZAHLEN

Darstellung negativer Zahlen

Für die Darstellung negativer Zahlen in Rechnern werden vier verschiedene Formate benutzt:

- Darstellung mit Betrag und Vorzeichen (B+V)

- Stellenkomplement-Darstellung (Einerkomplement-Darstellung)

- Zweierkomplement-Darstellung

- Offset-Dual-Darstellung / Exzess-Darstellung

Darstellung mit Betrag und Vorzeichen (B+V)

Eine Stelle wird als Vorzeichenbit benutzt.

- MSB = Most Significant Bit

Das am weitesten links stehende Bit ist das Vorzeichen

- MSB = 0 ➡ positive Zahl
- MSB = 1 ➡ negative Zahl

Beispiel:

- 0001 0010 = +18
- 1001 0010 = -18

Nachteile:

- Bei Addition und Subtraktion müssen die Vorzeichen der Operanden gesondert betrachtet werden.
- Es gibt zwei Repräsentationen der Zahl 0
  - Mit positivem und mit negativem Vorzeichen

Stellenkomplement / Einerkomplement

Stellenkomplement der entsprechenden positiven Zahl.

Um eine Zahl zu negieren, wird jedes Bit der Zahl komplementiert.

Dies entspricht dem **Einerkomplement**:

- n ist die Anzahl der Bitstellen, z.B. $n=4$ ➡ 4-Bit-Zahlen

$$z_{ek} = (2^n - 1) - z$$

Beispiel:

$4_{10} = 0100_2$ ➡ $-4_{10} = 1011_{ek}$

$-4_{10} = (2^4 - 1) - 4 = 11_{10} = 1011_2$

Negative Zahlen sind wiederum durch ein gesetztes Bit in der ersten Stelle (MSB) charakterisiert.

Einerkomplement-Darstellung

Vorteil gegenüber der Darstellung mit Vorzeichenbit:

- Die erste Stelle bei Addition und Subtraktion muss nicht gesondert betrachtet werden.

Aber

- Es gibt weiterhin zwei Darstellungen der Null

Zweierkomplement-Darstellung

Vermeidung dieses Nachteils:

- Man addiert nach der Stellenkomplementierung noch eine 1

Man erhält so das Zweierkomplement:

$$z_{zk} = 2^n - z$$

Komplementbildung

$0 \ldots 0$ ➡ $1 \ldots 1_{ek}$

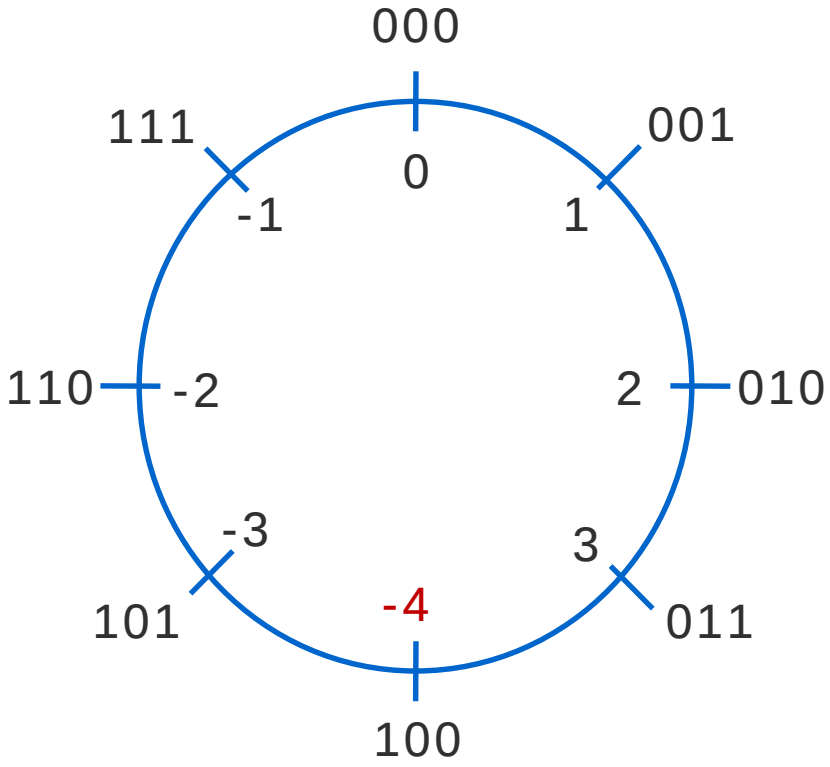
➡ $0 \ldots 0_{zk}$

Zweierkomplement-Darstellung

Nachteil:

- Unsymmetrischer Zahlenbereich. Die kleinste negative Zahl ist betragsmäßig um 1 größer als die größte positive Zahl

Beispiel: 3-Bit-ZK-Zahlen



Zweierkomplement-Darstellung

Alle anderen negativen Zahlen werden um 1 verschoben, das MSB bleibt aber gleich 1.

Aus der ersten Stelle kann das Vorzeichen der Zahl abgelesen werden

Aus dieser Konstruktion ergibt sich der Stellenwert des MSB einer Zweierkomplementzahl mit $n+1$ Bit zu -2^n:

$z_n\ z_{n-1}$ ... z_0 hat den Wert:

$$Z = -z_n 2^n + z_{n-1} 2^{n-1} + \ldots + z_0$$

Darstellung negativer Zahlen –Beispiel

Die Zahl -77_{10} soll mit 8 Bit dargestellt werden

$77_{10} = 0100\ 1101_2$

Mit Vorzeichenbit : $-77 = 1100\ 1101_2$

Einerkomplement : $-77 = 1011\ 0010_2$

Zweierkomplement : $-77 = 1011\ 0011_2$

Bitweise komplementieren

Addition von 1

Offset-Dual- (Exzess-)Darstellung

Wird hauptsächlich bei der Exponenten-Darstellung von Gleitkommazahlen benutzt.

Die Darstellung einer Zahl erfolgt in Form ihrer **Charakteristik**.

Der gesamte Zahlenbereich wird durch Addition einer Konstanten (Exzess, Offset) so nach oben verschoben, dass die kleinste (negative) Zahl die Darstellung **0…0** erhält.

Bei n Stellen ist der **Offset** 2^{n-1}

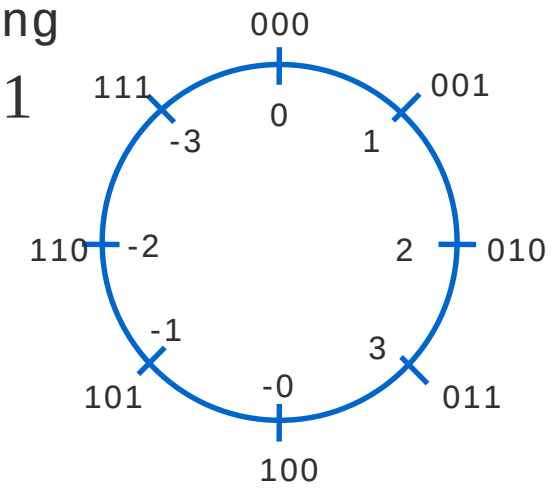
- Beispiel: $n=8$ ➡ Offset 128

Der Zahlenbereich ist hier auch **asymmetrisch**.

Zusammenfassung der Möglichkeiten

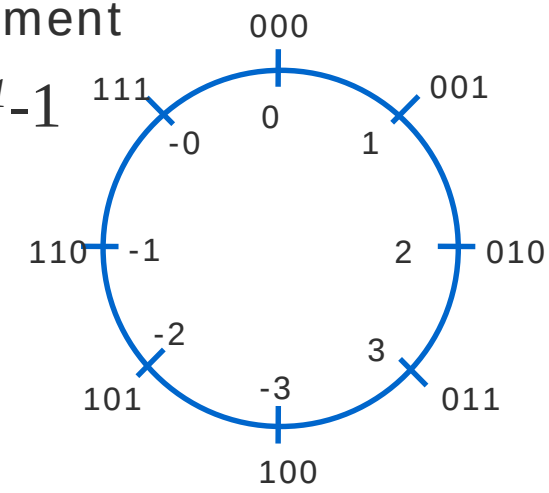
B+V-Darstellung

$-(2^{n-1}-1) : 2^{n-1}-1$



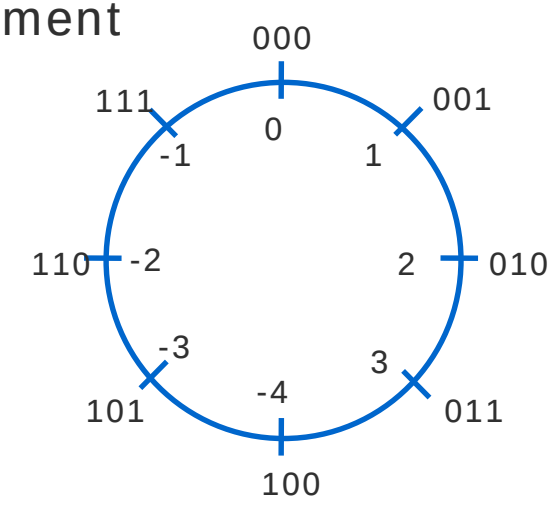
Einerkomplement

$-(2^{n-1}-1) : 2^{n-1}-1$



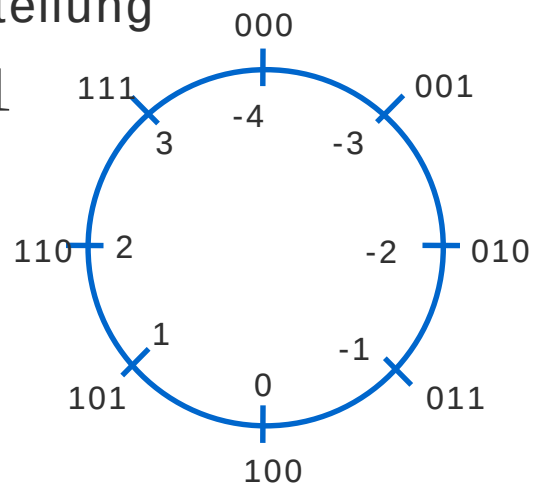
Zweierkomplement

$-2^{n-1} : 2^{n-1}-1$



Exzessdarstellung

$-2^{n-1} : 2^{n-1}-1$



Zusammenfassung der Möglichkeiten

Darstellung mit				
Dezimalzahl	Betrag + Vorzeichen	Einer-komplement	Zweier-komplement	Charakteristik
-4	- - -	- - -	1 0 0	0 0 0
-3	1 1 1	1 0 0	1 0 1	0 0 1
-2	1 1 0	1 0 1	1 1 0	0 1 0
-1	1 0 1	1 1 0	1 1 1	0 1 1
0	1 0 0, 0 0 0	1 1 1, 0 0 0	0 0 0	1 0 0
1	0 0 1	0 0 1	0 0 1	1 0 1
2	0 1 0	0 1 0	0 1 0	1 1 0
3	0 1 1	0 1 1	0 1 1	1 1 1

REELLE ZAHLEN (FEST- UND GLEITKOMMAZAHLEN)

Fest- und Gleitkommazahlen

Zahlendarstellung auf dem Papier:

- Ziffern 0 .. 9
- Vorzeichen + -
- Komma ,

Zahlendarstellung im Rechner:

- Binärziffern 0, 1

➡ spezielle Vereinbarungen für die Darstellung von Vorzeichen und Komma im Rechner sind erforderlich

Darstellung des Vorzeichens: wurde im vorigen Abschnitt behandelt

Darstellung des Kommas: 2 Möglichkeiten

- Festkommadarstellung (Fixed-point)
- Gleitkommadarstellung (Floating point)

Festkomma-Zahlen

Vereinbarung

- Das Komma sitzt innerhalb des Maschinenwortes, das eine Dualzahl enthalten soll, an einer **festen Stelle**.
- Meist setzt man das Komma hinter die letzte Stelle.

Eigenschaften

- Andere Zahlen können durch entsprechende Maßstabsfaktoren in die gewählte Darstellungsform überführt werden.
- Negative Zahlen: meist Zweierkomplement-Darstellung.
- Festkomma-Darstellungen werden heute hardwareseitig nicht mehr verwendet, jedoch bei Ein- oder Ausgabe!

Festkomma-Zahlen

Datentyp "integer" (Ganzzahlen) ist ein spezielles Festkommaformat.

Manche Programmiersprachen erlauben die Definition von Ganzzahlen unterschiedlicher Länge.

Größe (Bit)	Typische Namen	Vorzeichen	Grenzen des Wertebereichs (Zweierkomplement)	
			min	max
8	char, Byte/byte, modern: **int8_t** bzw. **uint8_t**	signed	−128	127
		unsigned	0	255
16	Word, Short/short, Integer, modern: **int16_t** bzw. **uint16_t**	signed	−32.768	32.767
		unsigned	0	65.535
32	DWord/Double Word, int, long (Windows auf 16/32/64-Bit Systemen; Unix/Linux auf 16/32-Bit Systemen), modern: **int32_t** bzw. **uint32_t**	signed	−2.147.483.648	2.147.483.647
		unsigned	0	4.294.967.295
64	Int64, QWord/Quadword, long long, Long/long (Unix/Linux auf 64-Bit Systemen), modern: **int64_t** bzw. **uint64_t**	signed	−9.223.372.036.854.775.808	9.223.372.036.854.775.807
		unsigned	0	18.446.744.073.709.551.615
128	Int128, Octaword, Double Quadword	signed	$\approx -1{,}70141 \cdot 10^{38}$	$\approx 1{,}70141 \cdot 10^{38}$
		unsigned	0	$\approx 3{,}40282 \cdot 10^{38}$

Quelle: Wikipedia

GLEITKOMMAZAHLEN

ZUNÄCHST ABSTRAKT
ACHTUNG: COMPUTER VERWENDEN KONKRET IEEE-P 754-FLOATING-POINT-STANDARD

Gleitkomma-Darstellung

Zur Darstellung von Zahlen, die betragsmäßig sehr groß oder sehr klein sind, verwendet man die Gleitkommadarstellung.

Sie entspricht einer halblogarithmischen Form

$$X = \pm \text{Mantisse} \cdot b^{\text{Exponent}}$$

Die Basis b ist für eine bestimmte Gleitkomma-Darstellung fest (meist 2 oder 16) und braucht damit nicht mehr explizit repräsentiert zu werden.

Gleitkommazahlen werden meist nicht im Zweierkomplement, sondern mit Betrag und Vorzeichen dargestellt.

Gleitkomma-Darstellung

Bei der Mantisse ist die Lage des Kommas wieder durch Vereinbarung festgelegt (meist links vom MSB).

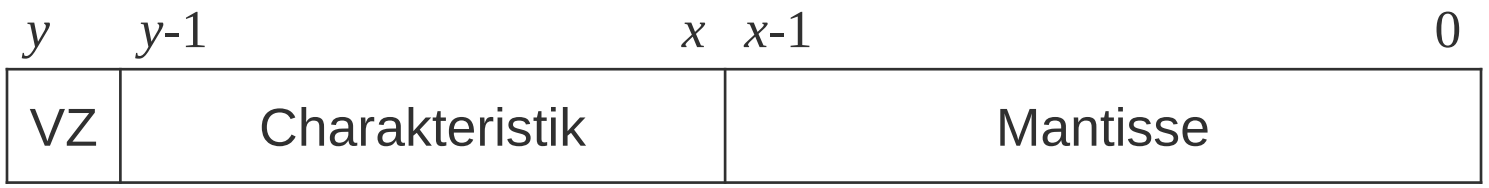
Der Exponent ist eine ganze Zahl, die in Form ihrer **Charakteristik** dargestellt wird.

Sowohl für die Charakteristik als auch für die Mantisse
wird im Rechner eine feste Anzahl von Speicherstellen festgelegt.

Die Länge der Charakteristik y-x bestimmt die Größe des Zahlenbereichs.

Die Länge der Mantisse x bestimmt die Genauigkeit der Darstellung.

Gleitkomma-Maschinenformat



$$\text{Dezimalzahl} = (-1)^{\text{VZ}} \times (0,\text{Mantisse}) \times b^{\text{Exponent}}$$

$$\text{Exponent} = \text{Charakteristik} - b^{(y-1)-x}$$

Normalisierung

Eine Gleitkommazahl heißt **normalisiert**, wenn für den Wert der Mantisse gilt:

$$\frac{1}{b} \leq \text{Mantisse} < 1$$

In dualer Darstellung ist die erste Stelle nach dem Komma gleich 1, d.h. (0,**1** . . . . .)

Ausnahme:
Bei der Zahl 0 sind alle Stellen der Mantisse gleich Null.

Normalisierung

Legt man für die Zahl 0 ein spezielles Bitmuster fest, ist die erste Stelle der Mantisse in normalisierter Form immer gleich 1.

Die erste Stelle der Mantisse braucht im Maschinenformat gar nicht erst dargestellt zu werden, d.h. (0,1 . . . . .)

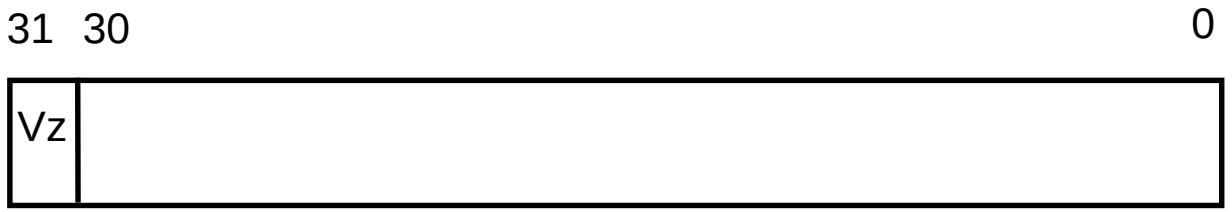
Man spart ein Bit bei der Speicherung oder gewinnt bei gleichem Speicherbedarf ein Bit an Genauigkeit.

Bei arithmetischen Operationen und bei der Konversion in andere Darstellungen darf diese Stelle natürlich nicht vergessen werden.

Beispiel: Darstellung der Zahl 7135_{10}

3 verschiedene Maschinenformate mit je 32 Bit und b = 2
Die Zahl 7135_{10} wird in jedem dieser Formate dargestellt.

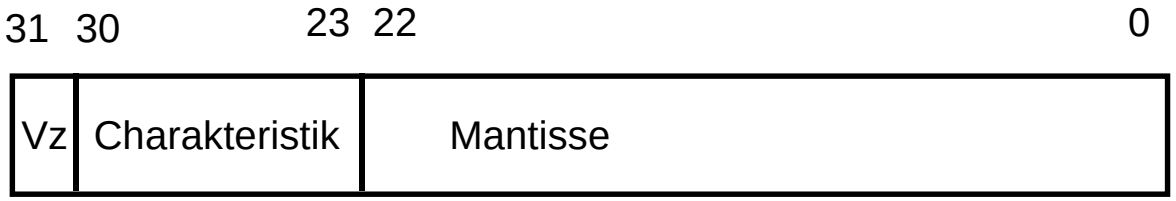
a) Festkommadarstellung mit Zweierkomplement



0 000 0000 0000 0000 0001 1011 1101 1111_2 = 0000 $1BDF_{16}$

Beispiel: Darstellung der Zahl 7135_{10}

b) Gleitkommadarstellung, normalisiert:



$0\ 100\ 0110\ 1\ 110\ 1111\ 0111\ 1100\ 0000\ 0000_2$ = $46EF\ 7C00_{16}$

c) Gleitkommadarstellung, normalisiert, erste "1" implizit:

$0\ 1000\ 1101\ \ 101\ 1110\ 1111\ 1000\ 0000\ 0000_2$ = $46DE\ F800_{16}$

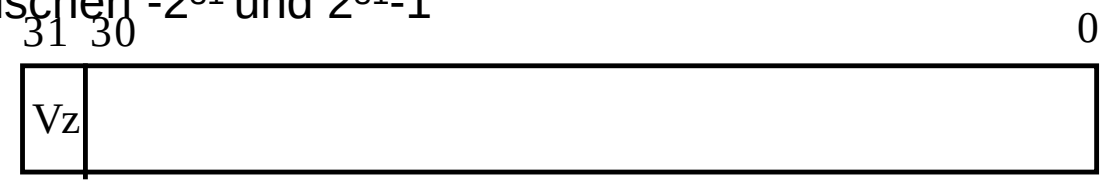
Darstellbarer Zahlenbereich

Die Anzahl darstellbarer Zahlen (Bitkombinationen) ist zwar in allen drei Fällen gleich (2^{32})

Der Bereich und damit die Dichte darstellbarer Zahlen auf dem Zahlenstrahl ist aber sehr unterschiedlich.

Darstellbarer Zahlenbereich

Format a) Zahlen zwischen -2^{31} und $2^{31}-1$



Format b)

negative Zahlen: $-(1-2^{-23}) \cdot 2^{127}$ ... $-0{,}5 \cdot 2^{-128}$

positive Zahlen: $0{,}5 \cdot 2^{-128}$ ... $(1-2^{-23}) \cdot 2^{127}$

und Null

Darstellbarer Zahlenbereich

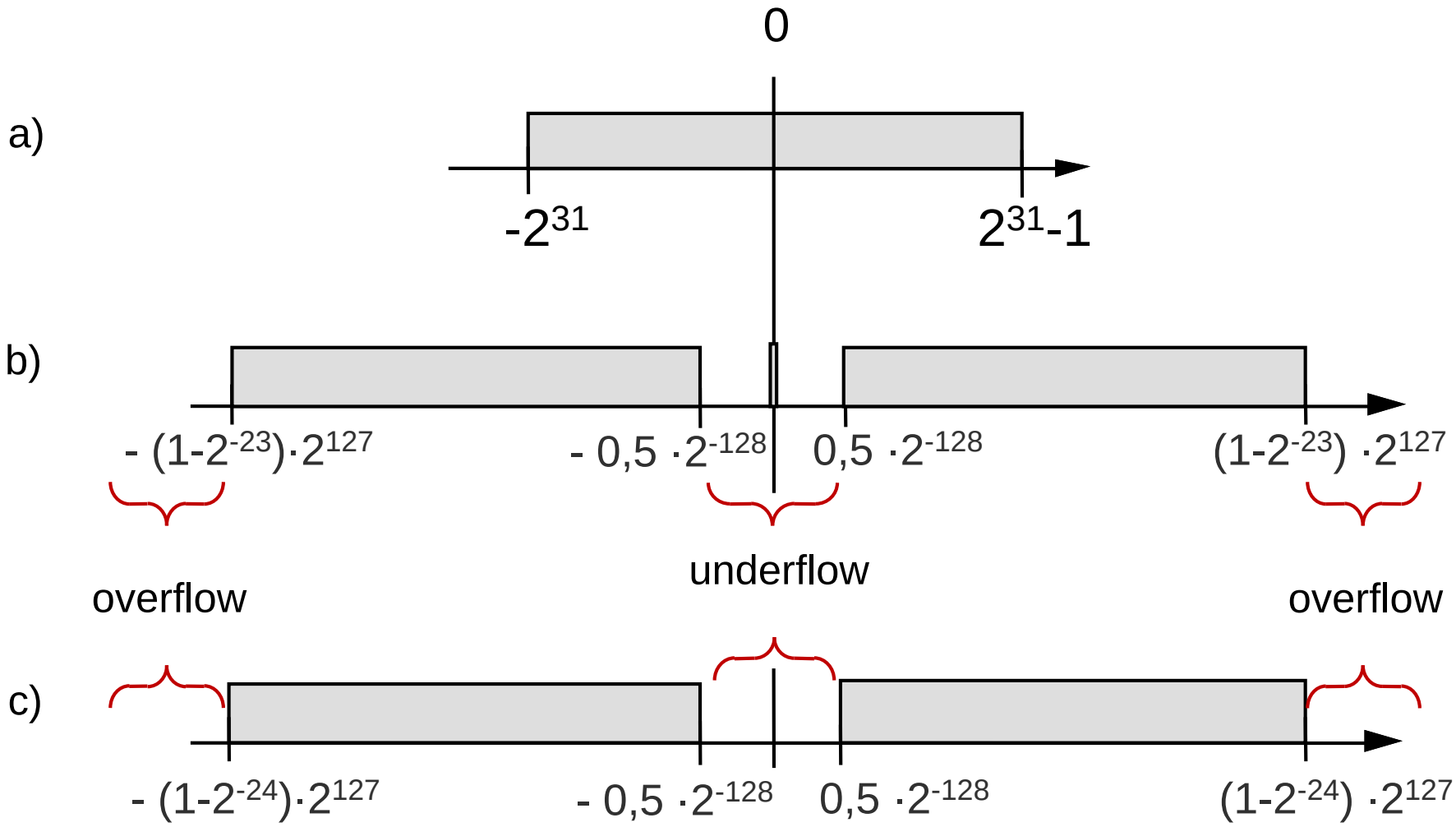
Format c) normalisierte Gleitkommadarstellung

negative Zahlen: $-(1-2^{-24})\cdot 2^{127}$ ... $-0{,}5\cdot 2^{-128}$

positive Zahlen $0{,}5\cdot 2^{-128}$ ... $(1-2^{-24})\cdot 2^{127}$

Die Null kann nicht dargestellt werden!

Darstellbarer Zahlenbereich



Charakteristische Zahlen

Um verschiedene Gleitkommadarstellungen miteinander vergleichen zu können, definiert man drei charakteristische Zahlen:

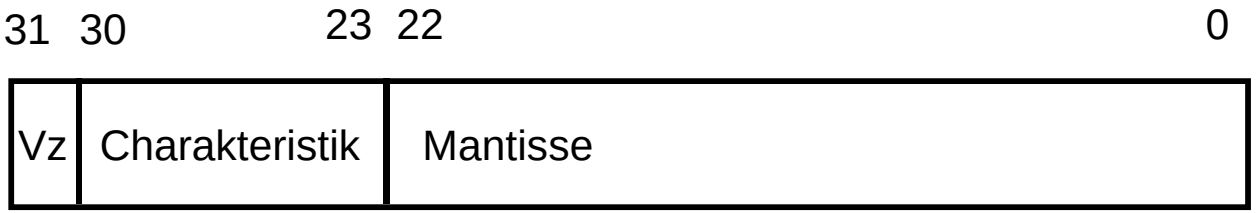
- maxreal ist die größte darstellbare normalisierte positive Zahl

- minreal ist die kleinste darstellbare normalisierte positive Zahl

- smallreal ist die kleinste Zahl, die man zu 1 addieren kann, um einen von 1 verschiedenen Wert zu erhalten.

Charakteristische Zahlen – Beispiel

In Format b) im letzten Beispiel



maxreal = $(1 - 2^{-23}) \cdot 2^{127}$ minreal = $0{,}5 \cdot 2^{-128}$

Die Zahl 1 wird normalisiert als $0{,}5 \cdot 2^1$ dargestellt.

Die nächstgrößere darstellbare Zahl hat in der Mantisse zusätzlich zur 1 in Bit 22 eine 1 in Bit 0.

smallreal = $0{,}00000000000000000000001_2 \cdot 2^1$,
also smallreal = $2^{-23} \cdot 2^1 = 2^{-22}$

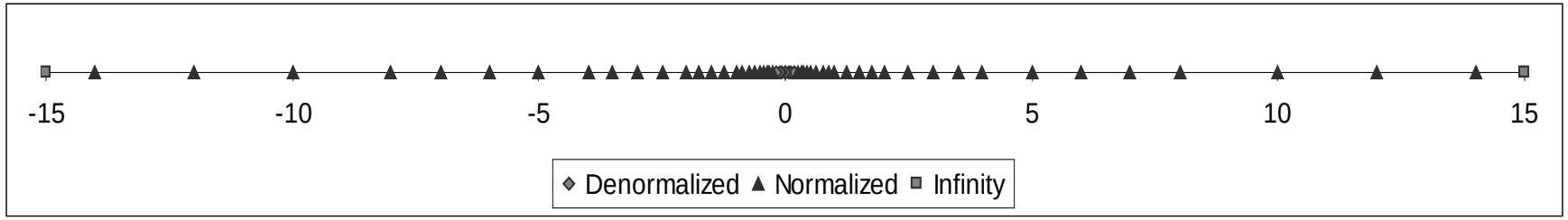
Ungenauigkeiten

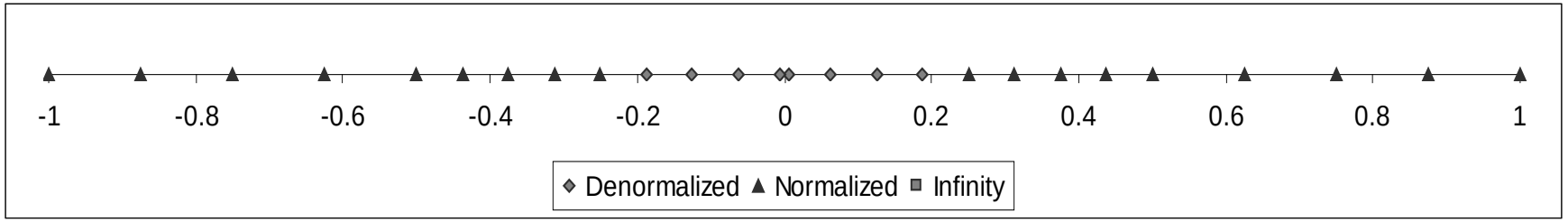
Die Differenz zwischen zwei aufeinanderfolgenden Zahlen wächst bei Gleitkomma-Zahlen exponentiell mit der Größe der Zahlen, während sie bei Festkomma-Zahlen konstant ist.

Bei der Darstellung großer Zahlen ergibt sich damit auch eine hohe Ungenauigkeit.

Die Gesetzmäßigkeiten, die für reelle Zahlen gelten, werden für Maschinendarstellungen verletzt!
(auch wenn diese Zahlen in einer höheren Programmiersprache oft **`real`** heißen).

Ungenauigkeiten





Source: Computer Systems: A Programmer's Perspective

Beispiel

Das Assoziativgesetz $(x + y) + z = x + (y + z)$ gilt selbst dann nicht unbedingt, wenn kein overflow oder underflow auftritt.

z.B.: $x = 1; y = z = \text{smallreal}/2$

$$
\begin{aligned}
(x + y) + z &= (1 + \text{smallreal}/2) + \text{smallreal}/2 \\
&= 1 + \text{smallreal}/2 \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
x + (y + z) &= 1 + (\text{smallreal}/2 + \text{smallreal}/2) \\
&= 1 + \text{smallreal} \\
&\neq 1
\end{aligned}
$$

IEEE-P 754-FLOATING-POINT-STANDARD

Normierung (IEEE-Standard)

IEEE-P 754-Floating-Point-Standard

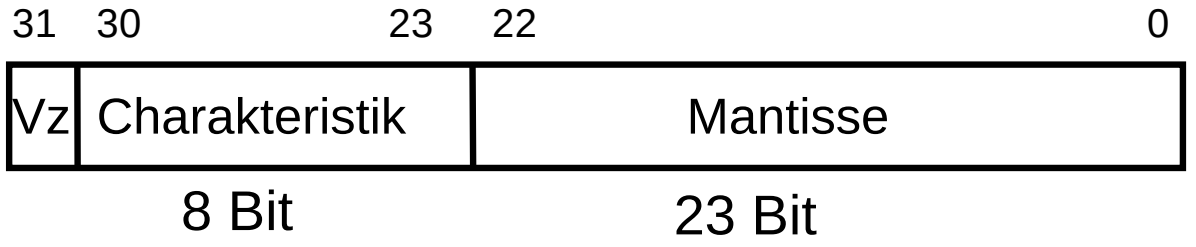
In vielen Programmiersprachen lassen sich Gleitkomma-Zahlen mit verschiedener Genauigkeit darstellen

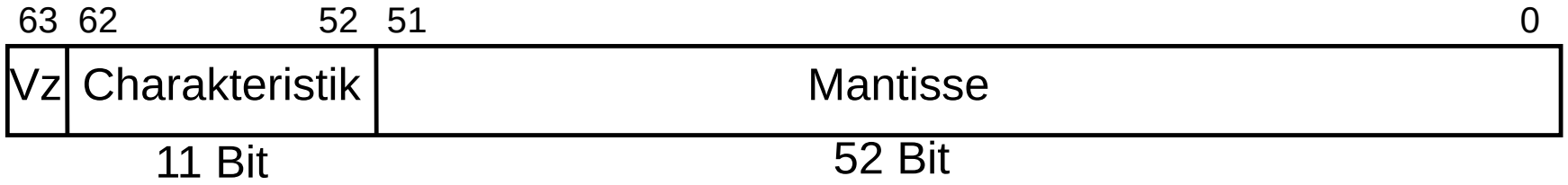
- z.B. in C: float
  double
  long double

Der IEEE-Standard definiert mehrere Darstellungsformen

- IEEE single: 32 Bit
  IEEE double: 64 Bit
  IEEE extended: 80 Bit

IEEE-P 754-Floating-Point-Standard





Maschinenformate des IEEE-Standards

Eigenschaften des IEEE-P 754

Die Basis b ist gleich 2.

Das erste Bit der Mantisse wird implizit zu 1 angenommen, wenn die Charakteristik nicht nur Nullen enthält.

Normalisierung: das erste Bit der Mantisse (die implizite 1) steht vor dem Komma.

Ist die Charakteristik gleich 0, entspricht dies dem gleichen Exponenten wie die Charakteristik 1.

Das erste Bit der Mantisse wird aber dann explizit dargestellt

Auch die Null ist darstellbar

Eigenschaften des IEEE-P 754

Sind alle Bits der Charakteristik gleich 1, signalisiert dies eine Ausnahmesituation.

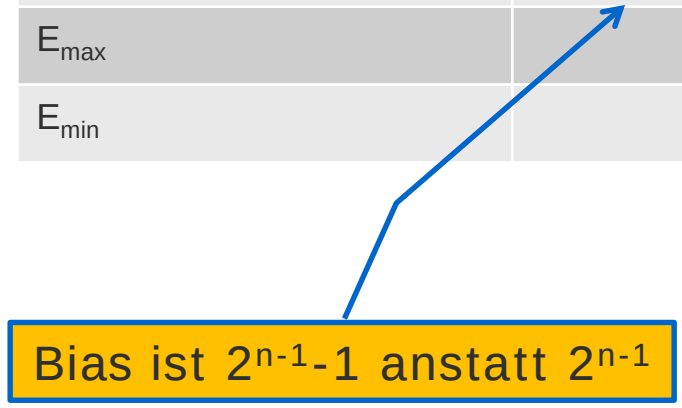
Wenn zusätzlich die Mantisse gleich Null ist, wird die Situation „overflow“ (bzw. die „Zahl“ $\pm \infty$) kodiert.

Dies erlaubt es dem Prozessor, eine Fehlerbehandlung einzuleiten.

Intern arbeiten Rechner nach dem IEEE-Standard mit 80 Bit, um Rundungsfehler unwahrscheinlicher zu machen.

Zusammenfassung der Parameter des IEEE-P 754

Parameter	Single	Double
Bits Gesamt	32	64
Bits Mantisse	23(+1)	52(+1)
Bits Charakteristik	8	11
Exponent Bias	+127	+1023
E_{max}	+127	+1023
E_{min}	-126	-1022



Zusammenfassung des 64-Bit-IEEE-Formats

Charakter.	Zahlenwert	Bemerkung
0	$(-1)^{Vz}$ 0,Mantisse · 2^{-1022}	
1	$(-1)^{Vz}$ 1,Mantisse · 2^{-1022}	
…	$(-1)^{Vz}$ 1,Mantisse · $2^{\text{Charakteristik} - 1023}$	
2046	$(-1)^{Vz}$ 1,Mantisse · 2^{1023}	
2047	Mantisse = 0: $(-1)^{Vz} \infty$	Overflow
2047	Mantisse ≠ 0: NaN	Not a number

Literatur

IEEE Computer Society:

- IEEE Standard for Binary Floating-Point Arithmetic ANSI/IEEE Standard 754-1985, SIGPLAN Notices, Vol. 22, No. 2, pp 9-25, 1978

D. Goldberg:

- What every computer scientist should know about floating point arithmetic, ACM Computing Surveys, Vol. 13, No. 1, pp. 5-48,1991

Rundung

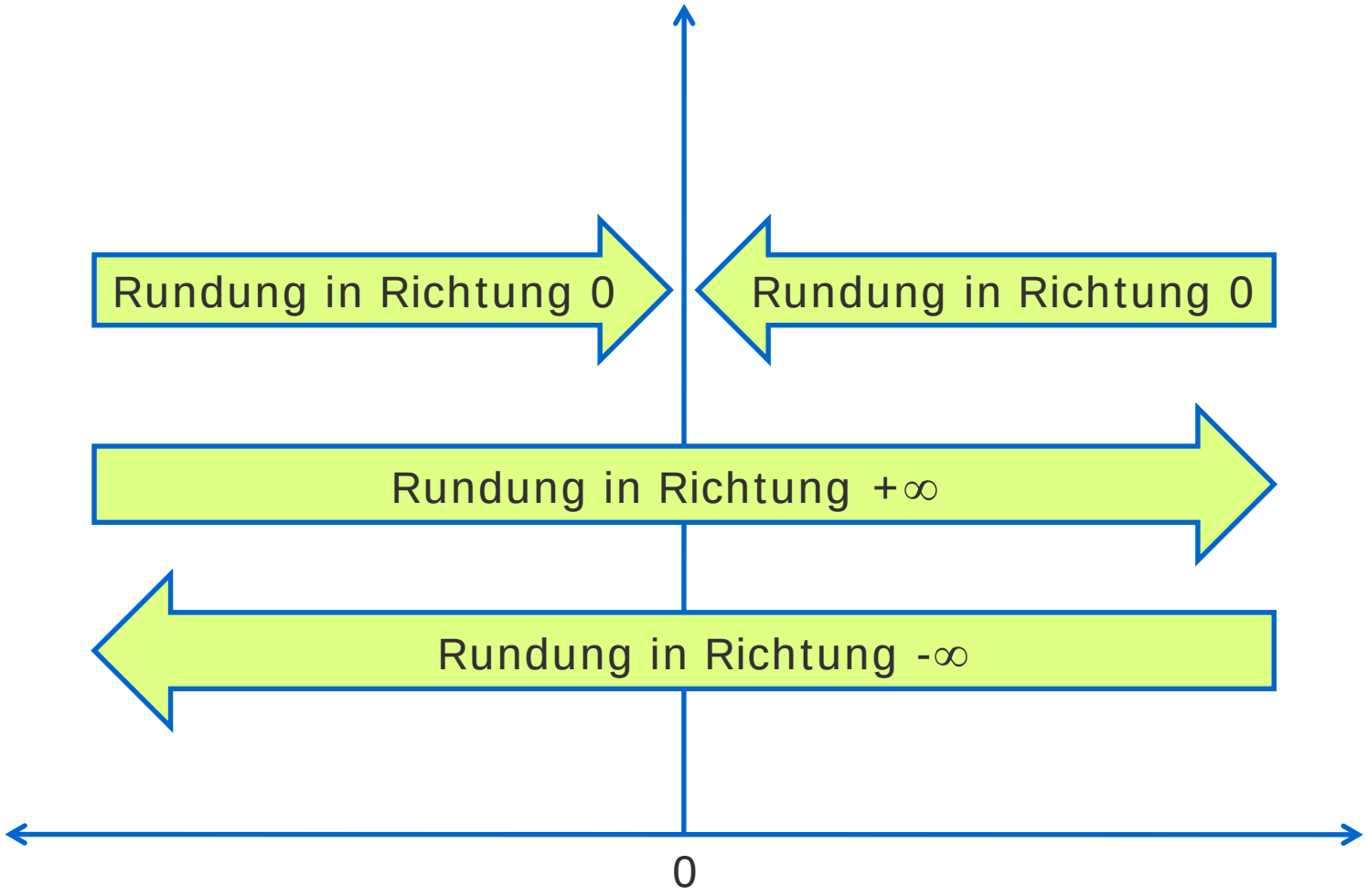
IEEE Standard Forderung:

- Das Ergebnis, das man durch eine arithmetische Operation mit dem Rechner erhält, soll dasselbe sein, als wenn man exakt rechnet und anschließend entsprechend eines geeigneten Modus rundet.

IEEE Standard definiert vier Rundungsmodi:

- Rundung zum nächstliegenden Gleitkommawert:
  - Falls der Abstand zu zwei Gleitkommawerten gleich ist, wird zu jenem Wert gerundet, dessen niederwertigste Stelle eine gerade Ziffer ist („round-to-even“-Regel)
- Rundung zum nächsten Gleitkommawert in Richtung 0
- Rundung zum nächsten Gleitkommawert in Richtung $+\infty$
- Rundung zum nächsten Gleitkommawert in Richtung $-\infty$

Rundung


Rundung in Richtung 0
Rundung in Richtung 0
Rundung in Richtung +∞
Rundung in Richtung -∞
0

Rundung

Am schwierigsten zu implementierende Rundung:

- Rundung zum nächstliegenden Gleitkommawert

Eine Möglichkeit: Summe exakt berechnen und anschließend runden.

- sehr lange Register, sehr aufwändig

Auch mit weniger Hardware-Aufwand möglich?

Es gibt zwei Fälle für eine Rundung bei Addition und Subtraktion:

- auftretender Übertrag
- Exponentenanpassung

Beispiel a: Übertrag bei Addition

Basis 10, drei signifikante Stellen

	$2{,}34 \times 10^2$
(Übertrag)	$+8{,}51 \times 10^2$
	$10{,}85 \times 10^2$
wird gerundet zu	$1{,}08 \times 10^3$

Beispiel b: ungleiche Exponenten

Basis 10, drei signifikante Stellen

Exponenten-anpassung

$$
\begin{array}{rr}
2{,}34 \times 10^{2} & 2{,}34 \times 10^{2} \\
+2{,}56 \times 10^{0} & +0{,}0256 \times 10^{2} \\
\hline
 & 2{,}3656 \times 10^{2}
\end{array}
$$

wird gerundet zu $2{,}37 \times 10^{2}$

Beispiel c: Übertrag und ungl. Exponenten

Basis 10, drei signifikante Stellen

(beides)

$$
\begin{array}{rl}
 9{,}51 & \times 10^2 \\
+0{,}642 & \times 10^2 \\
\hline
10{,}152 & \times 10^2
\end{array}
$$

wird gerundet zu $1{,}02 \times 10^3$

Problem

Für jeden dieser Fälle muss die Summe mit mehr als drei signifikanten Stellen berechnet werden, um eine korrekte Rundung zu ermöglichen.

Es gibt auch Fälle, bei denen eine Rechnung mit mehr als drei signifikanten Stellen notwendig ist, obwohl keine Rundung erfolgt

- Subtraktion nahe beieinanderliegender Zahlen
- Siehe Beispiel d

Beispiel d: Subtraktion von naheliegenden Zahlen

$$
\begin{array}{rl}
 1{,}47 & \times 10^2 \\
-0{,}876 & \times 10^2 \\
\hline
 0{,}594 & \times 10^2
\end{array}
$$

Bei den bisherigen Beispielen reichte eine zusätzliche Stelle aus.

Es gibt aber auch Fälle, bei denen dies nicht genügt.

Beispiel e:

$$\begin{array}{rl} 1{,}01 & \times 10^2 \\ -0{,}0376 & \times 10^2 \\ \hline 0{,}9724 & \times 10^2 \end{array}$$

wird gerundet zu $\mathbf{0{,}972 \times 10^2}$

Wenn die niederwertigste Ziffer 6 von 0,0376 gestrichen würde, wäre das Ergebnis 0,973 anstatt 0,972.

Rundungs- und Prüfstelle

Es lässt sich zeigen, dass unter Vernachlässigung der "round-to-even"-Regel zwei weitere Stellen für eine korrekte Rundung stets ausreichend sind.

Diese beiden Stellen heißen:

- die Rundungsstelle **r** und
- die Prüfstelle **g**.

Aber: Die "round-to-even"-Regel erfordert zusätzlichen Aufwand.

Beispiel f:

Fünf signifikante Stellen:

```
4,5674 ×10^0          4,5674
2,5001 ×10^-4       + 0,00025001
                    -------------
                      4,56765001
                            gr
```

wird gerundet zu **4,5677**

Rundungsstelle r und **Prüfstelle g** genügen nicht

Information: sind alle niederwertigeren Stellen hinter der Rundungsstelle gleich Null, dann nur ein sticky-Bit

Sticky-Bit

Für eine richtige Rundung ist die Information ausreichend, ob alle niederwertigeren Stellen hinter der Rundungsstelle gleich Null sind.

Es genügt ein Bit: "sticky"-Bit

Wenn eine der Stellen, die durch das Angleichen der Exponenten beider Operanden gestrichen werden, ungleich Null ist, wird das "sticky"-Bit gesetzt.

Falls das Ergebnis in gleichem Abstand zum oberen und unteren nächstliegenden Fließkommawert liegt, entscheidet das "sticky"-Bit, ob nach oben oder nach unten gerundet wird.

ADDITION UND SUBTRAKTION

Addition und Subtraktion

Schaltungen zur Addition von Festkomma-Dualzahlen:

- Grundlage für die Durchführung aller arithmetischen Verknüpfungen

Denn:

- Subtraktion ≙ Addition der negativen Zahl
- $X - Y = X + (-Y)$

Multiplikation und Division lassen sich ebenfalls auf die Addition zurückführen.

- Bei Gleitkommazahlen:
  - Mantisse und Exponent werden separat verarbeitet.
  - Hierbei bildet die Addition von Festkomma-Dualzahlen die Grundlage.

Grundtypen von Addierern sind wichtig

Vom Halbaddierer zum Volladdierer

Bei der Addition zweier Dualzahlen: Summe und Übertrag entstehen als Ergebnis

Funktionstabelle:

a	b	s	ü
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

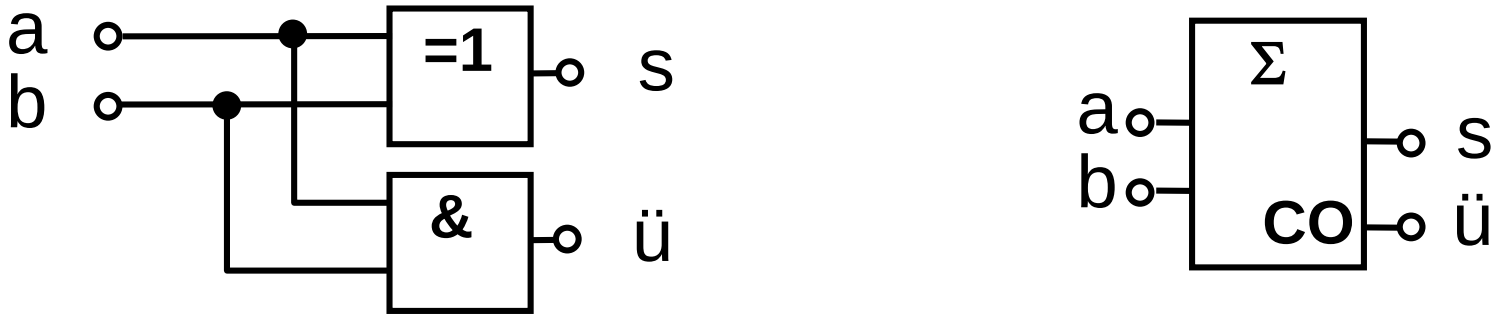
Man nennt dies einen Halbaddierer

Halbaddierer

Gleichungen: $s = a\,\bar{b} \vee \bar{a}\,b = a \oplus b$

$\ddot{u} = a\,b$

Das Schaltbild und das Schaltsymbol:



Schaltbild und Schaltsymbol eines 1-Bit-Halbaddierers

Mehrstellige Dualzahlen

Zusätzlicher Eingang für den Übertrag der vorhergehenden Stellen ist nötig.

a_i	b_i	$ü_i$	s_i	$ü_{i+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

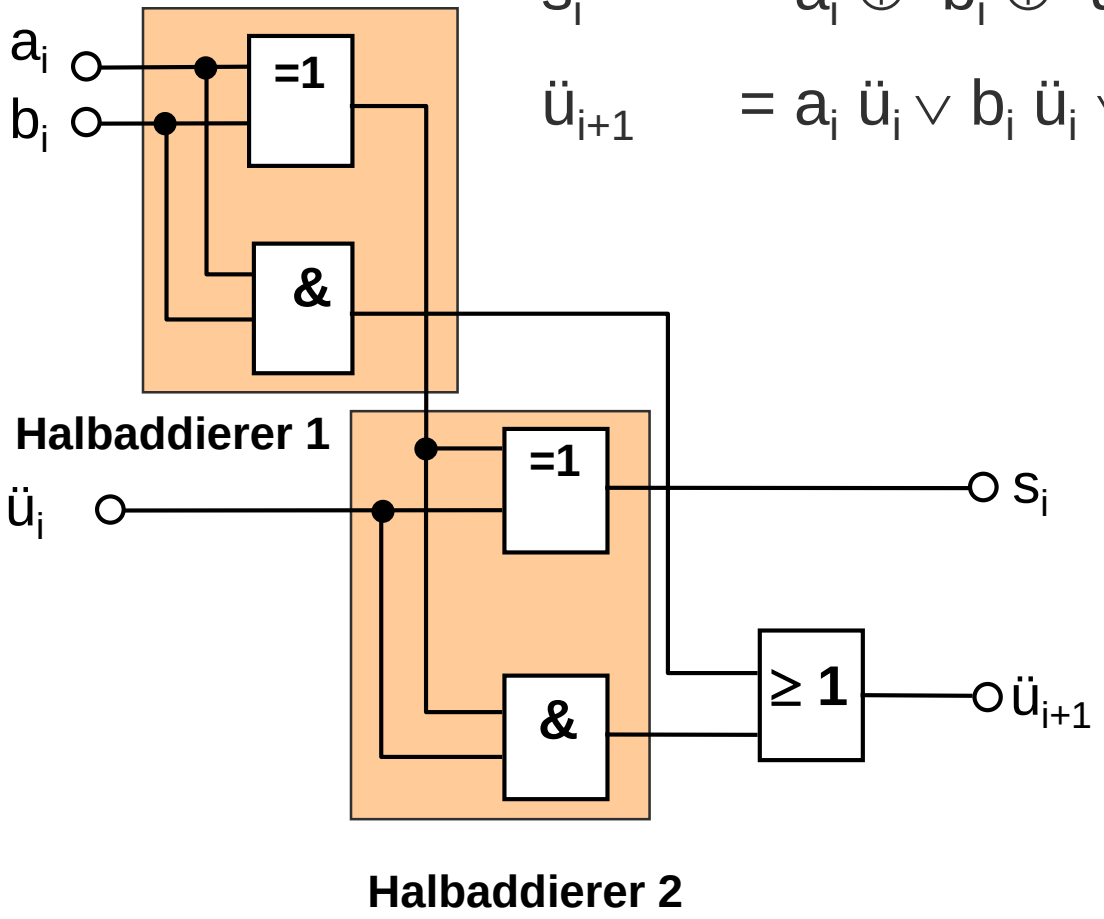
Dies nennt man einen Volladdierer

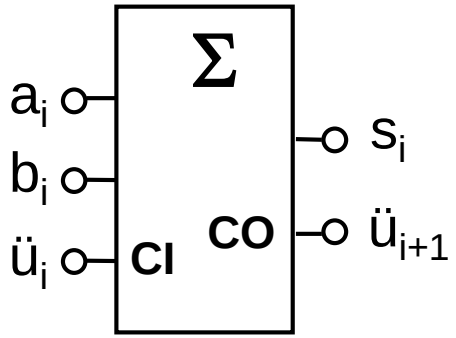
Gleichungen, Schaltnetz und Schaltsymbol

Ausgangsgleichungen:

$$s_i = a_i \oplus b_i \oplus \ddot{u}_i$$

$$\ddot{u}_{i+1} = a_i\,\ddot{u}_i \vee b_i\,\ddot{u}_i \vee a_i\,b_i = (a_i \oplus b_i)\,\ddot{u}_i \vee a_i\,b_i$$





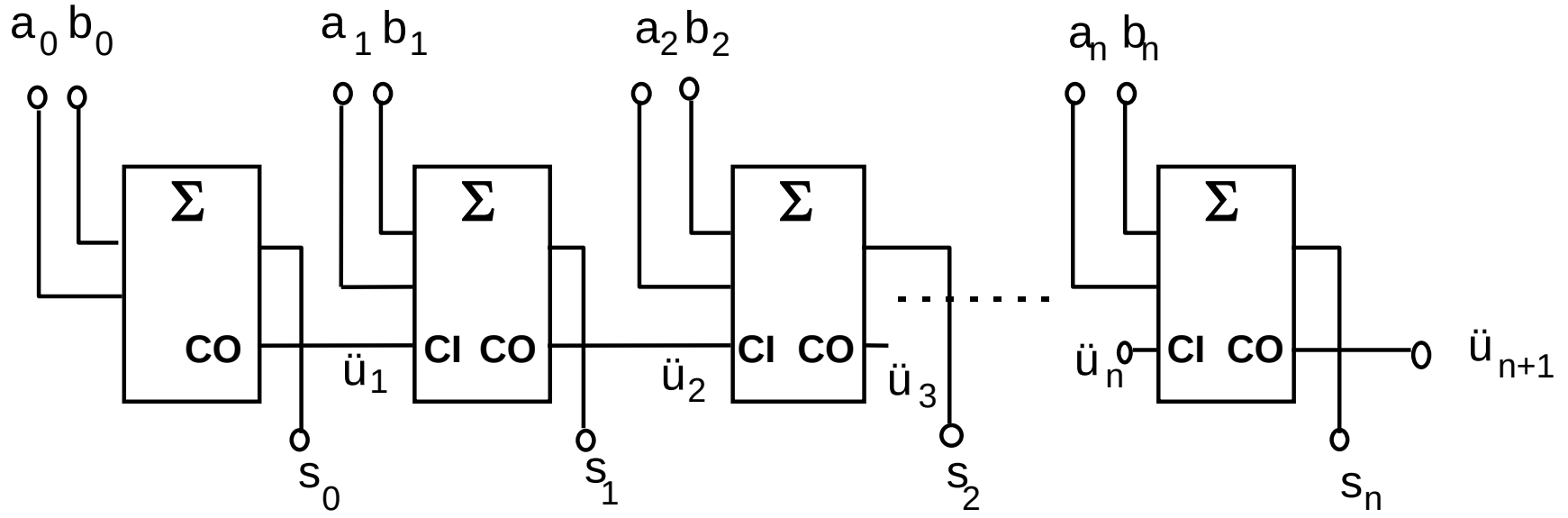
Volladdierer

Carry-ripple-Addierer

Addieren zweier Dualzahlen mit mehreren Stellen

Einfachste Lösung:

- Für jede Stelle einen Volladdierer vorsehen und den Übertrag der Stelle i im Volladdierer der Stelle i+1 berücksichtigen.
- Die Stelle geringster Wertigkeit (LSB, least significant bit) kann mit einem Halbaddierer realisiert werden.



Probleme

Ergebnis der Addition einer Stelle ist erst dann gültig, wenn der Übertrag aus der vorhergehenden Stelle berechnet ist.

Ungünstiger Fall: das Durchlaufen durch alle Stufen muss abgewartet werden (Carry-ripple-Addierer, to ripple = rieseln).

Die Stabilisierungsdauer ist proportional zur Anzahl der Stellen.

Man nennt den Carry-ripple-Addierer auch Asynchroner Parallel-Addierer, da er bit-parallel addiert, d.h. alle Bits der Operanden gleichzeitig benutzt.

Carry-lookahead-Addierer

Um den Nachteil der großen Additionszeit des Carry-ripple-Addierers zu vermeiden:

- Alle Überträge direkt aus den Eingangsvariablen bestimmen (Carry-Lookahead)

Es gilt:

- $ü_{i+1} = a_i\, b_i \vee (a_i \oplus b_i)\, ü_i = g_i \vee p_i\, ü_i$
- $s_i = (a_i \oplus b_i) \oplus ü_i = p_i \oplus ü_i$

mit

- $g_i = a_i\, b_i$ (generate carry, erzeuge Übertrag) und
- $p_i = (a_i \oplus b_i)$ (propagate carry, leite Übertrag weiter)

- g_i und p_i können direkt aus den Eingangsvariablen erzeugt werden.

Berechnung der Überträge aus den Eingangsvariablen

Die rekursive Berechnung der $\ddot{u}_i$ kann aufgelöst werden, indem sukzessive die Ausdrücke für die Berechnung des Übertrags in den vorhergehenden Stellen eingesetzt werden.

$$\ddot{u}_{i+1} = g_i \vee p_i \ddot{u}_i$$

Man erhält

$$\ddot{u}_1 = g_0 \vee p_0 \ddot{u}_0$$

$$\ddot{u}_2 = g_1 \vee p_1 g_0 \vee p_1 p_0 \ddot{u}_0$$

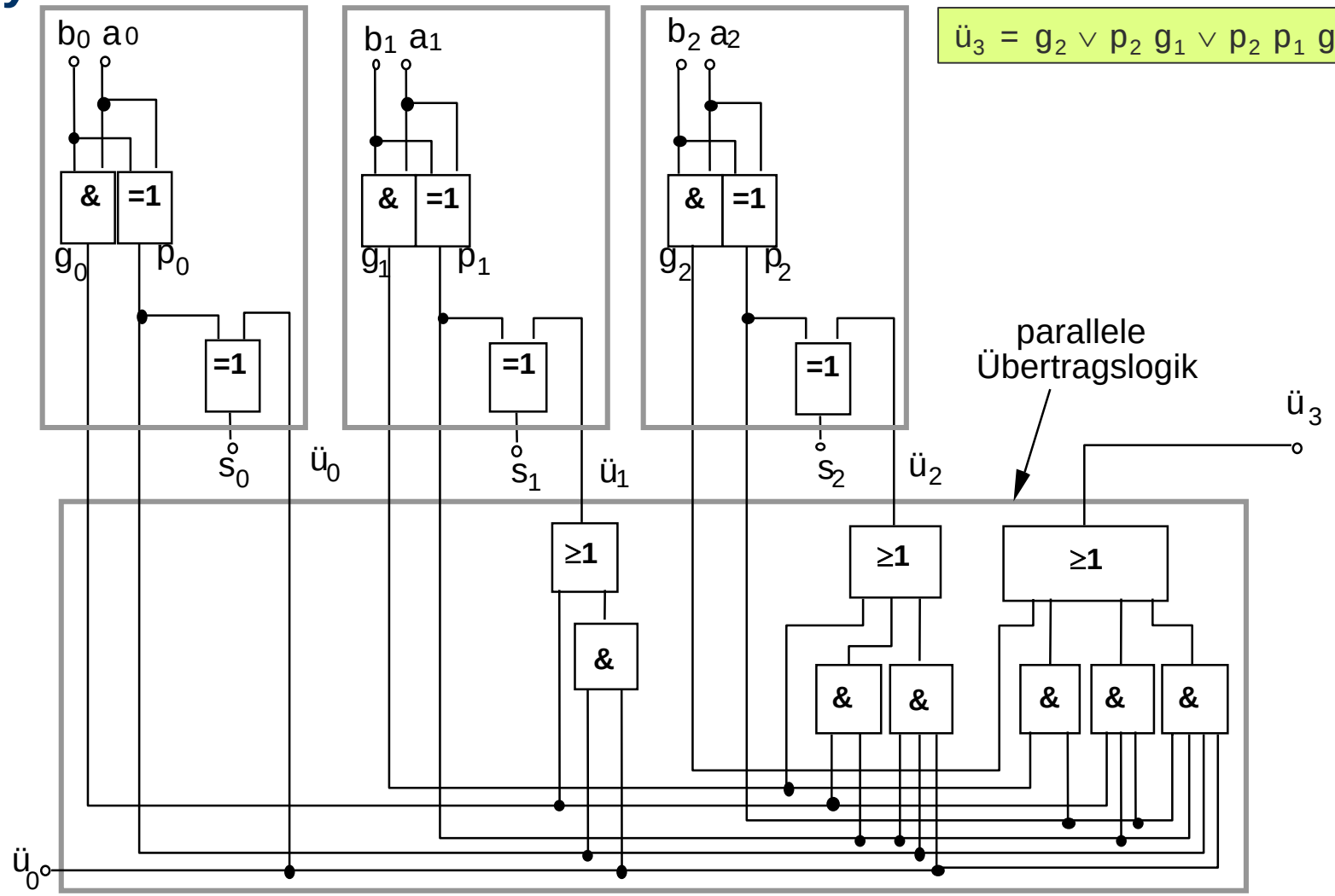
$$\ddot{u}_3 = g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 \ddot{u}_0$$

usw.

Die Additionszeit wird damit weitgehend unabhängig von der Stellenzahl, weil die Berechnung des Übertrags in allen Stufen sofort (vorausschauend) beginnen kann. Deshalb wird dieser Addierer als **Carry-lookahead-Addierer** bezeichnet.

Schaltbild:
3-Bit-Carry-lookahead-Addierer

$$ü_3 = g_2 \vee p_2\, g_1 \vee p_2\, p_1\, g_0 \vee p_2\, p_1\, p_0\, ü_0$$



Carry-lookahead-Addierer

Problem:

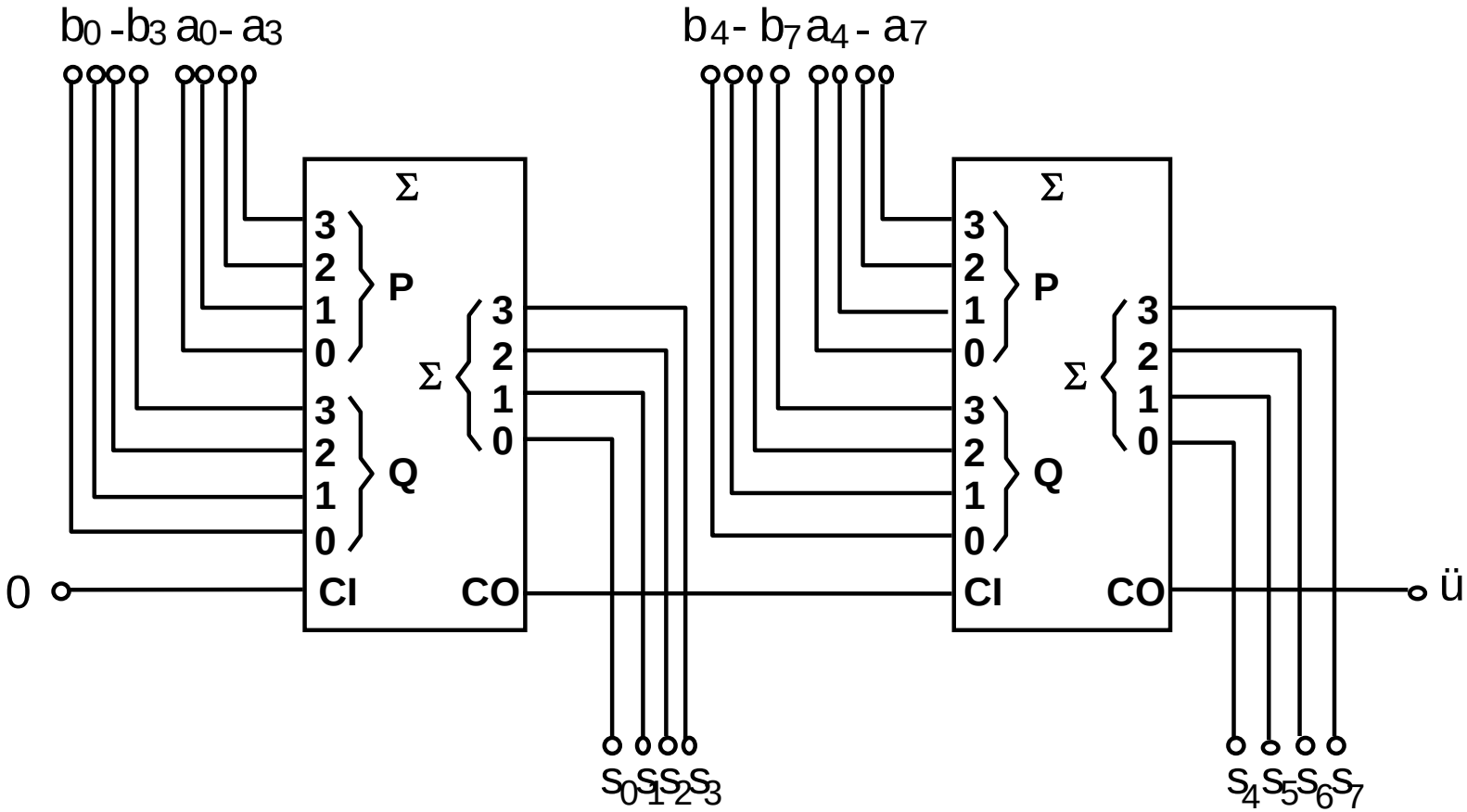
- Größe des Hardware-Aufwands steigt mit steigender Stellenzahl stark an.

Lösungen:

- kleinere Carry-lookahead-Addierer mit paralleler Übertragserzeugung, die seriell kaskadiert werden
- Blocküberträge der kleineren Blöcke parallel verarbeiten

➡Hierarchie von Carry-lookahead-Addierern

Carry-lookahead-Addierer



Kaskadierung zweier 4-Bit Carry-lookahead-Addierer zur Addition von 8-Bit-Zahlen

Addition und Subtraktion

SUBTRAKTION

Subtraktion

Subtraktion durch Addition des **Zweierkomplements**.

Zweierkomplement: bit-weise Komplementierung der Zahl und anschließende Addition von 1

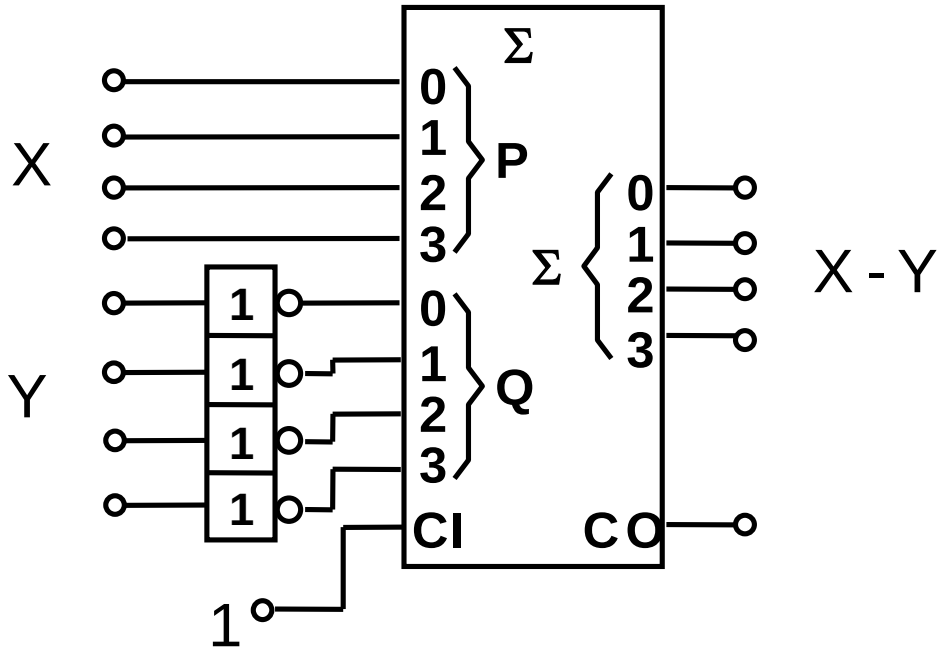
$$X - Y = X + (\overline{Y} + 1) = X + \overline{Y} + 1$$

Man beachte:

- Beide Eingabezahlen sind in der Zweierkomplement-Form gegeben.
- Am Ausgang entsteht wieder eine Zahl in Zweierkomplement-Form

Subtraktion

Die beiden Additionen können mit einem Addierer vorgenommen werden, indem man den Übertragseingang ausnutzt.

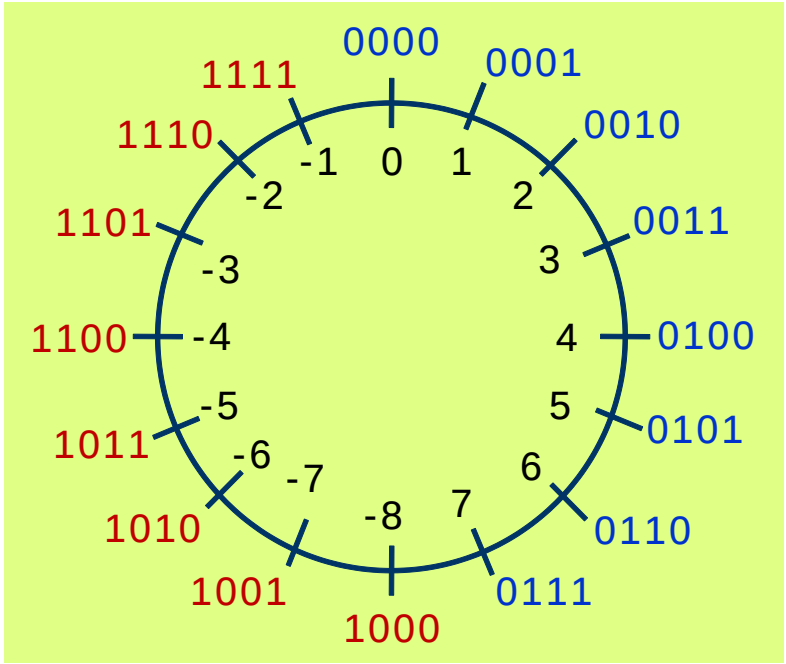


Subtraktion von Zweierkomplementzahlen

Sonderfälle

Bei der Addition lassen sich 3 Sonderfälle unterscheiden

1) Beide Summanden sind positiv
   - die Vorzeichenbits beider Zahlen sind 0.
   - das Ergebnis muss positiv sein.
   - Das Ergebnis ist nur dann korrekt, wenn sein Vorzeichenbit gleich 0 ist, ansonsten wurde der Zahlenbereich überschritten.
   - Man kann sich diese Situation anhand des Zahlenkreises klarmachen.




```
   5          0 1 0 1
+  2       +  0 0 1 0
-------    ------------
   7       0 0 0 0
           ---------
           0 1 1 1
```

Überträge gleich
Ergebnis korrekt

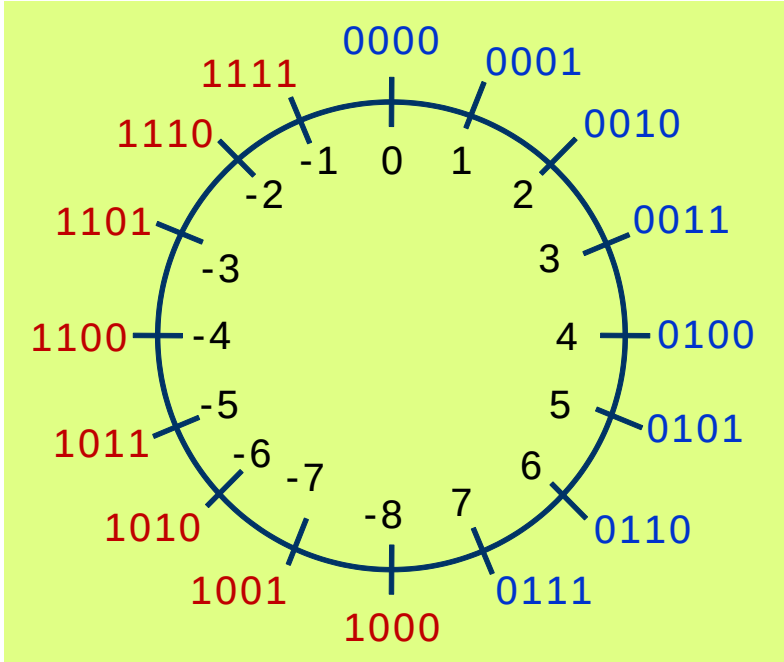
```
   5          0 1 0 1
+  6       +  0 1 1 0
-------    ------------
  11       0 1 0 0
           ---------
           1 0 1 1
```

Überträge ungleich
Überlauf
Ergebnis falsch

Sonderfall 2

2)Beide Summanden sind negativ

- Die Vorzeichenbits beider Zahlen haben den Wert 1.
- Das Ergebnis muss negativ sein.
- Das Ergebnis ist nur dann korrekt, wenn das Vorzeichenbit des Ergebnisses 1 ist.
- Die beiden vordersten Überträge müssen den gleichen Wert haben.

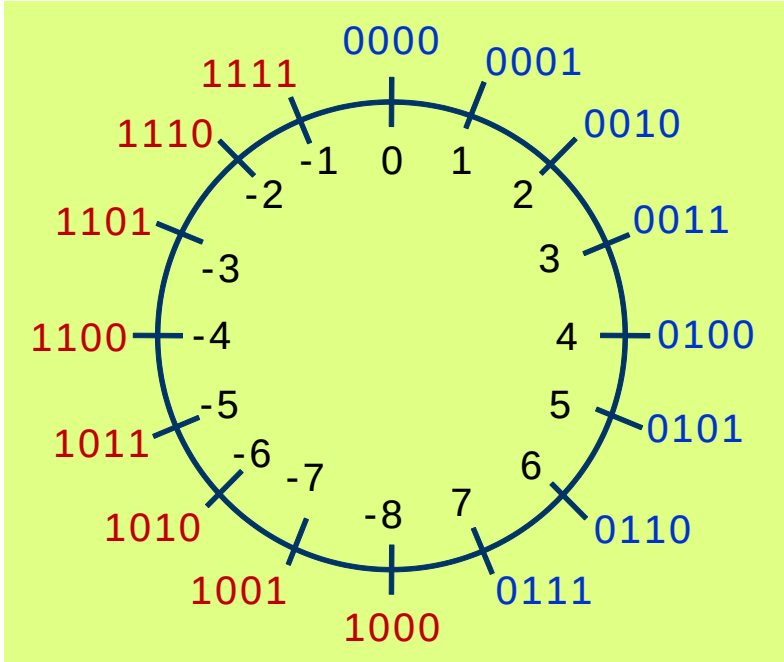


- 5	1 0 1 1	
+ (- 2)	+ 1 1 1 0	
	1 1 1 0	
= - 7	1 0 0 1	Überträge gleich Ergebnis korrekt

- 5	1 0 1 1	
+ (- 6)	+ 1 0 1 0	
	1 0 1 0	
= - 11	0 1 0 1	Überträge ungleich Überlauf Ergebnis falsch

Sonderfall 3

3) Beide Summanden haben unterschiedliche Vorzeichen:
- Das Ergebnis ist auf jeden Fall korrekt, das Vorzeichen hängt davon ab, ob Subtrahend oder Minuend betragsmäßig größer ist.
- Der Übertrag aus der vordersten Stelle ist zu streichen.




```
   5        0 1 0 1
+ (- 6)     1 0 1 0
-------    ---------
 = - 1     0 0 0 0
           1 1 1 1
```

Überträge gleich
Ergebnis korrekt

```
  - 5       1 0 1 1
+   6       0 1 1 0
-------    ---------
 =   1     1 1 1 0
           0 0 0 1
```

Überträge gleich
Ergebnis korrekt

Überlauferkennung

Allgemeine Überlauferkennung bei dualer Addition:

- korrekte Addition: beide Überträge sind gleich.

- Überlauf: beide Überträge sind ungleich.

Realisierung z.B. durch ein Antivalenzgatter (XOR)

GLEITKOMMA-ADDITION

Zusatzbetrachtung: Gleitkomma-Addition

Addition von zwei Gleitkommazahlen a_1 und a_2

$$a_1 = s_1 \times b^{e_1} \qquad a_2 = s_2 \times b^{e_2}$$

Beispiel: $a_1 = 3{,}21 \times 10^{2}$

$a_2 = 8{,}43 \times 10^{-1}$

Gerechnet wird mit zwei zusätzlichen Stellen in der Mantisse (Guard und Round) sowie dem Sticky-Bit.

Mantissenbits	G	R	S

Gleitkomma-Addition

1. Exponentenangleichung

- Gleitkommazahlen können nur addiert werden, wenn die Exponenten gleich sind

Schritt 1: Wenn $e_1 < e_2$, dann vertausche die Operanden, so dass gilt: $d = e_1 - e_2 \geq 0$

Schritt 2: Verschiebe die Mantisse s_2 um d Stellen nach rechts.

- Wenn d > 2, setze das Sticky Bit, falls die d-2 herausgeschobenen Stellen einen Wert $\neq 0$ ergeben.

Im Beispiel: d = 2 - (-1) = 3

$3{,}2100 \times 10^2$

$0{,}0084 \times 10^2$ Sticky-Bit gesetzt, da die 3 aus 8,43 herausgeschoben.

Gleitkomma-Addition

2. Mantissenaddition

- Addiere die beiden Mantissen

Im Beispiel:

$$\begin{array}{r} 3{,}2100 \times 10^2 \\ 0{,}0084 \times 10^2 \\ \hline 3{,}2184 \times 10^2 \end{array}$$

3. Normalisierung

- Normalisiere die entstandene Summe durch Verschieben der Mantisse und Korrektur des Exponenten.

Gleitkomma-Addition

4. Rundung

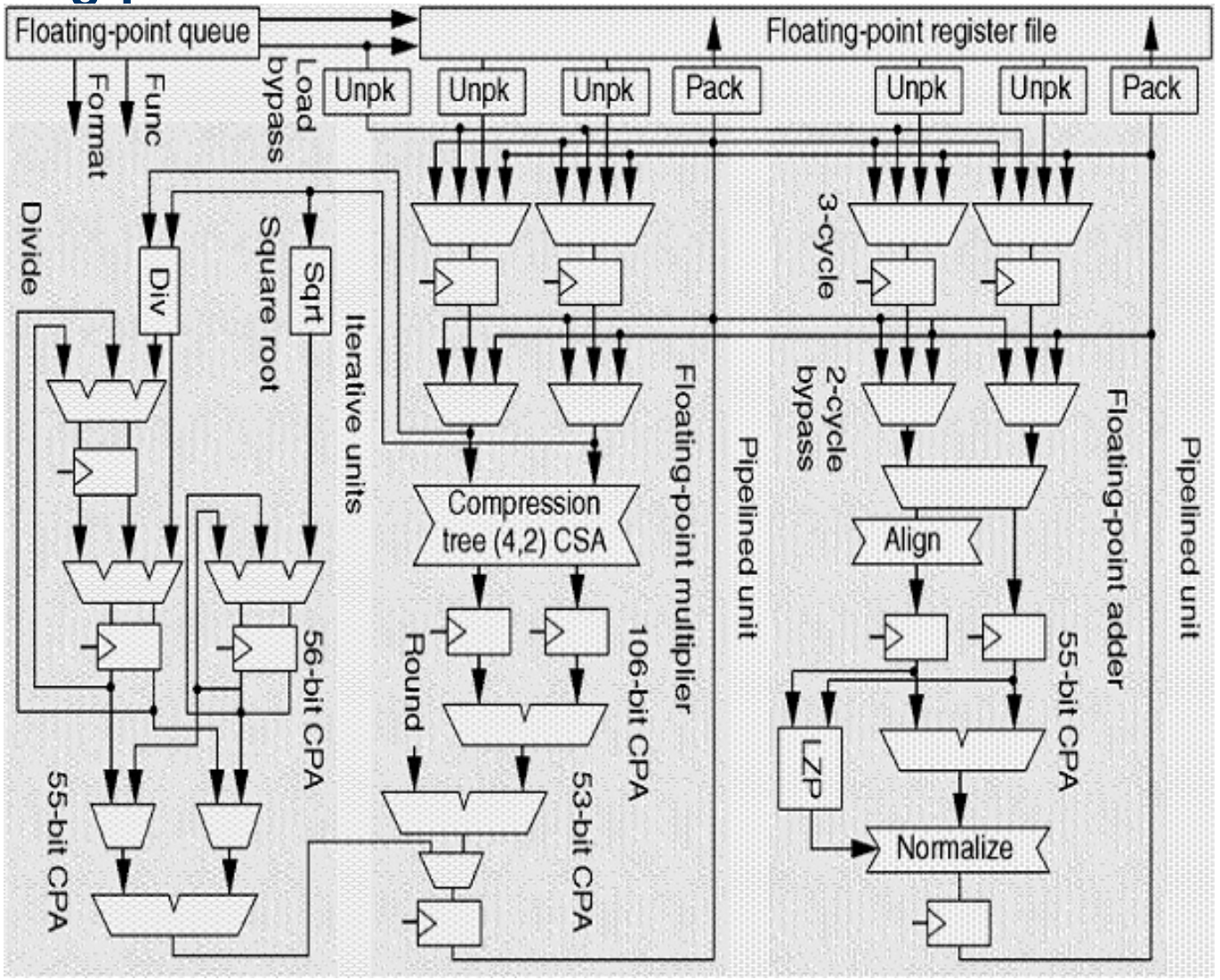
Runde unter Berücksichtigung der Stellen g, r und des Sticky-Bits, sowie der gegebenen Rundungsart (meist round to even)

Im Beispiel:

$$
\begin{array}{rl}
 & 3{,}2100 \times 10^2 \\
 & 0{,}0084 \times 10^2 \\
\hline
 & 3{,}2184 \times 10^2
\end{array}
$$

Ergebnis wird gerundet zu: $3{,}22 \times 10^2$

MIPS R10000 Floating-point Unit



See literature for multiplication and division!

ARITHMETISCH-LOGISCHE EINHEIT (ALU)

Arithmetisch-logische Einheit

Arithmetisch-logische Einheit (ALU, arithmetic logic unit):

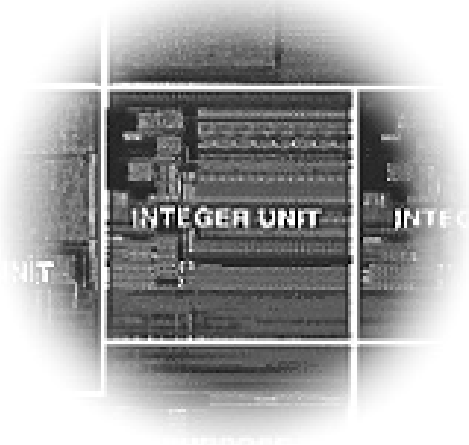
- Rechenwerk, der funktionale Kern eines Digitalrechners zur Durchführung logischer und arithmetischer Verknüpfungen.

Eingangsdaten der ALU:

- Daten und Steuersignalen vom Prozessor

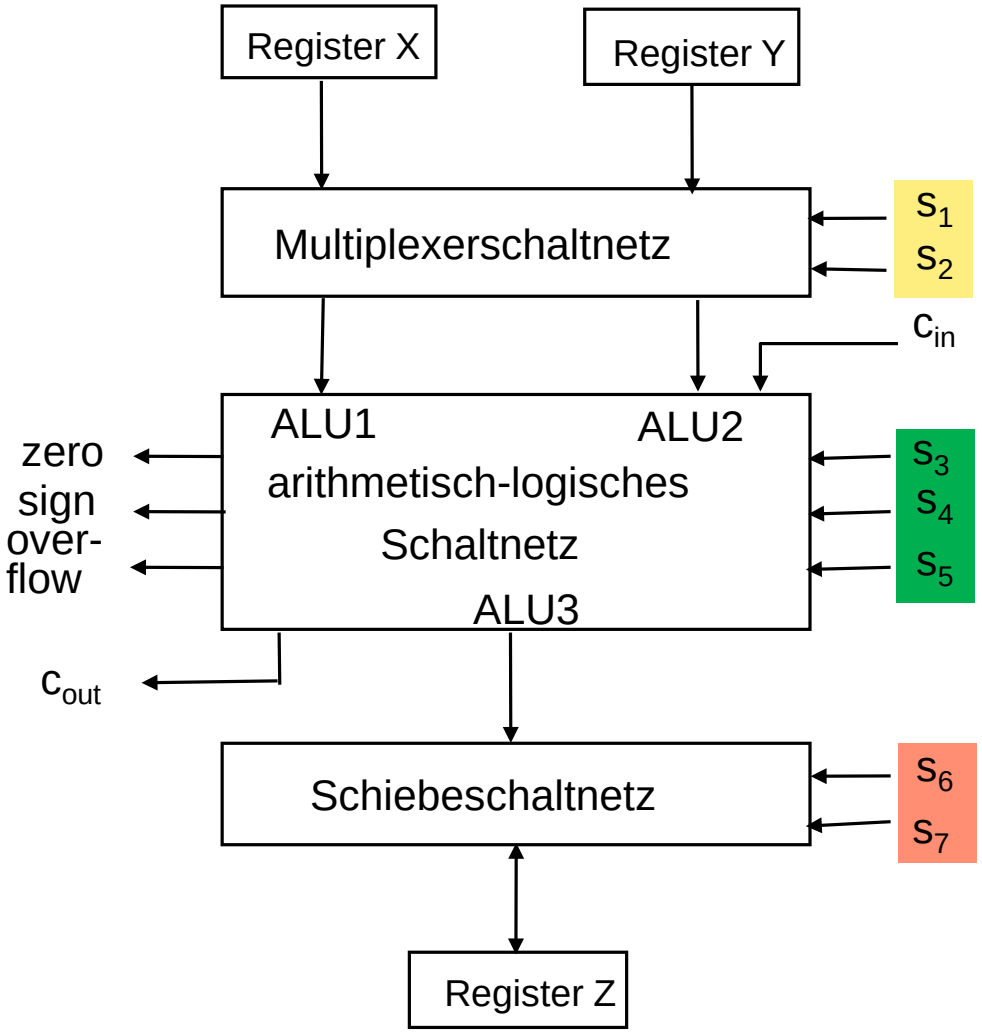
Ausgangsdaten der ALU:

- Ergebnisse und Statussignale an den Prozessor



Oft können die in einen Prozessor integrierten ALUs nur Festkommazahlen verarbeiten. Die Gleitkommaoperationen werden dann entweder von einer Gleitkommaeinheit ausgeführt oder per Software in eine Folge von Festkommabefehlen umgewandelt.

Schema einer einfachen ALU



s_1	s_2	ALU1	ALU2
0	0	X	Y
0	1	X	0
1	0	Y	0
1	1	Y	X

s_3	s_4	s_5	ALU3
0	0	0	ALU1 + ALU2 $+c_{in}$
0	0	1	ALU1 – ALU2 – Not(c_{in})
0	1	0	ALU2 – ALU1 – Not(c_{in})
0	1	1	ALU1 $\vee$ ALU2
1	0	0	ALU1 $\wedge$ ALU2
1	0	1	Not(ALU1) $\wedge$ ALU2
1	1	0	ALU1 $\oplus$ ALU2
1	1	1	ALU1 $\leftrightarrow$ ALU2

s_6	s_7	Z
0	0	ALU3
0	1	ALU3 ÷ 2
1	0	ALU3 × 2
1	1	Z speichern

Bestandteile der ALU

Registersatz

Multiplexerschaltnetz

Arithmetisch logisches Schaltnetz zur Durchführung arithmetisch logischer Operationen

Schiebeschaltnetz

Eingänge:

- Datenworte X und Y
- Steuersignale $s_1 \ldots s_7$ zur Festlegung der ALU-Operation

Ausgänge:

- Statussignale zero, sign und overflow
- Hiermit kann das Steuerwerk bestimmte ALU-Zustände erkennen und darauf entsprechend reagieren.

Beispiele

Einerkomplement von Y um ein Bit nach links verschoben in Z ablegen.

- Steuersignale: $s_1 \ldots s_7 = 10\ 111\ 10$
  - 10 : ALU1 = Y
  - 111: ALU3 = ALU1 ↔ ALU2
  - 10 : Z = ALU3 × 2

Ist $X > Y$?

- Statussignal "sign" bei der Operation $Y - X$.
- Steuersignale: $s_1 \ldots s_7 = 00\ 010\ 00$ und $c_{in} = 1$
  - 00 : ALU1 = X und ALU2 = Y
  - 010: ALU3 = ALU2 – ALU1 – not(c_{in})
  - 00 : Z = ALU3

Zusammenfassung

Zahlensysteme
- Positive Zahlen
- Negative Zahlen
- Umwandlung zwischen Zahlensystemen
- Reelle Zahlen (Fest- und Gleitkommazahlen)
  - In der Praxis zählt nur IEEE!
- Rundung von Zahlen

Schaltungen
- Addition / Subtraktion
- Multiplikation / Division

Arithmetisch-logische Einheit (ALU)
- Bestandteile einer ALU