Rechner Architektur I (RAI)

# Informationsverarbeitung

## Teil 2

Prof. Dr. Akash Kumar

*Chair for Processor Design*

# Rechenoperationen
# Arithmetic Operations

# Addition Beispiel



dezimal ($n = 2$)

$$\begin{array}{lll} a & & 5\ 9_{10} \\ b & + & 9\ 1_{10} \\ \hline s & = 1^1\ 5^1\ 0_{10} & \end{array}$$

dual ($n = 8$)

$$\begin{array}{l} 0\ 0\ 1\ 1\ 1\ 0\ 1\ 1_2 \\ +0\ 1\ 0\ 1\ 1\ 0\ 1\ 1_2 \\ \hline = 1^1\ 0^1\ 0^1\ 1^1\ 0^0\ 1^1\ 1^1\ 0_2 \end{array}$$

Schreibweise: Summe$^{\text{Übertrag der vorherigen Stelle}}$

Sukzessive stellenweise Addition der Summanden mit dem LSB beginnend bis hin zum MSB $(a_0 + b_0, \ldots, a_{n-1} + b_{n-1})$ zur Summe $(s_0, \ldots, s_{n-1})$ mit Übertragstechnik. Die bei der stellenweisen Addition auftretenden Überträge (carry) $(c_0, \ldots, c_{n-1})$ werden bei der Addition der jeweis nächstfolgenden Stelle mit dazu addiert. Der letzte Übertrag $c_{n-1}$ heisst auslaufender Übertrag (carry out).

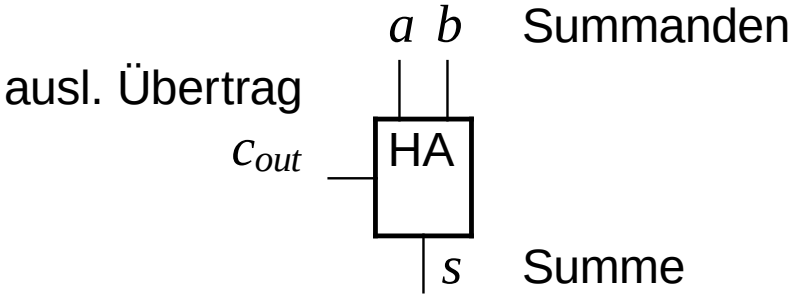
# Hardware-Realisierung der Addition



**1-Bit Addition**

**Halbaddierer (Half Adder (HA), ohne einlaufenden Übertrag)**

**Schaltsymbol**




Modulo 2 Addition ohne Übertrag

**Wertetabelle**

| $a$ | $b$ | $s$ | $c_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

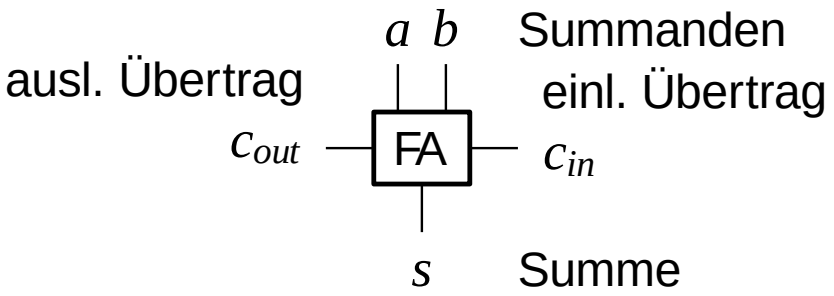
**Schaltfunktion**

$$s = a \wedge \bar{b} \vee \bar{a} \wedge b$$
$$= a \oplus b \quad (a \text{ xor } b)$$
$$c_{out} = a \wedge b$$

# Volladdierer (Full Adder (FA), mit einlaufenden Übertrag)



**Schaltsymbol**




Modulo 2 Addition mit Übertrag

**Wertetabelle**

| $c_{in}$ | $a$ | $b$ | $s$ | $c_{out}$ |
|---|---|---|---|---|
| 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 |

**Schaltfunktion**

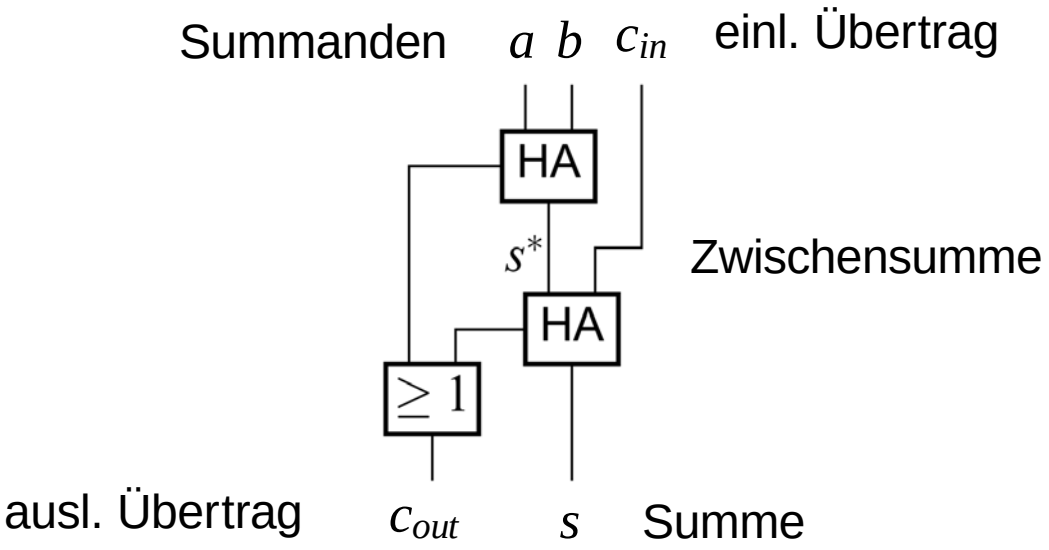
$$s = \overline{a} \wedge b \wedge \overline{c_{in}} \vee a \wedge \overline{b} \wedge \overline{c_{in}} \vee \overline{a} \wedge \overline{b} \wedge c_{in} \vee a \wedge b \wedge c_{in}$$

$$c_{out} = a \oplus b \oplus c_{in}$$

$$= a \wedge b \vee a \wedge c_{in} \vee b \wedge c_{in}$$

# Volladdierer, realisiert durch Halbaddierer

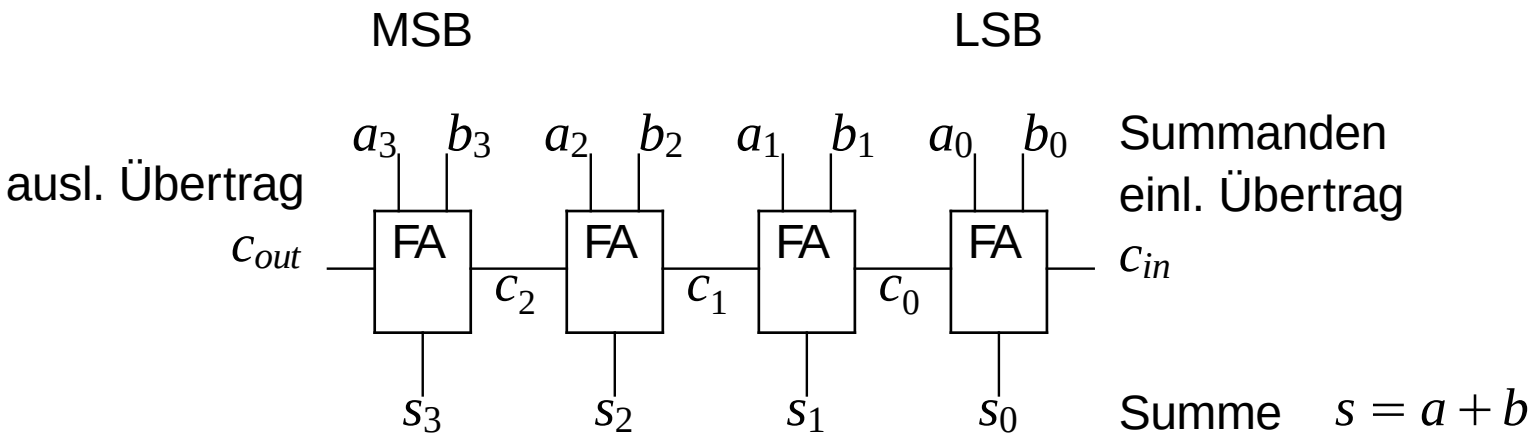





$$
\begin{aligned}
s^* &= a \wedge \overline{b} \vee \overline{a} \wedge b \\
&= a \oplus b \\
s &= s^* \wedge \overline{c}_{in} \vee \overline{s}^* \wedge c_{in} \\
&= s^* \oplus c_{in} \\
c_{out} &= a \wedge b \vee s^* \wedge c_{in} \qquad \text{(OR Gate)}
\end{aligned}
$$

# Beispiel: 4-Bit Addition (Ripple-Carry-Adder (RCA))






Addition zweier n-Bit Zahlen durch stellenweise Addition mit Übertrag mit dem LSB beginnend und Weiterreichen des Übertrages an die jeweils nachfolgende Stelle.

$$s_\nu = a_\nu \oplus b_\nu \oplus c_{\nu-1} \quad \text{für } \nu = 0, 1, \ldots, n-1$$

$$c_\nu = a_\nu \wedge b_\nu \vee a_\nu \wedge c_\nu \vee b_\nu \wedge c_\nu$$

$$c_{-1} = c_{in}$$

kann auch bei $c_{in} = 0$ entfallen

$$c_{out} = c_{n-1}$$

# Subtraktion (1) Beispiel



| dezimal ($n = 2$) | dual ($n = 8$) |
|---|---|
| $a \quad 9\ 1_{10}$ | $0\ 1\ 0\ 1\ 1\ 0\ 1\ 1_2$ |
| $b \quad -\ 5\ 9_{10}$ | $-\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 1_2$ |
| $d \quad = 3^1\ 2_{10}$ | $= 0^0\ 0^1\ 1^0\ 0^0\ 0^0\ 0^0\ 0^0\ 0_2$ |

Schreibweise: Differenz Übertrag (geborgt) der vorherigen Stelle

Sukzessive stellenweise Subtraktion mit dem LSB beginnend bis hin zum MSB mit Borgetechnik. Die bei der stellenweisen Subtraktion auftretenden (geborgten) Überträge (borrow) werden bei der Subtraktion der jeweis nächstfolgenden Stelle mit zum Subtrahenden addiert. Der letzte (geborgte) Übertrag $c_{n-1}$ heisst auslaufender (geborgter) Übertrag (borrow out).

# Subtraktion (2) Beispiel



dezimal ($n = 2$)

$$\begin{array}{rrrr} a & & 9 & 1_{10} \\ b & (- & 5 & 9_{10}\ ) \\ {}^{(10)}b & + & 4 & 1_{10} \\ \hline d & = {}^{1} & 3^{0} & 2_{10} \end{array}$$

dual ($n = 8$)

$$\begin{array}{rrrrrrrrr} & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1_2 \\ (- & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1_2\ ) \\ + & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1_2 \\ \hline = {}^{1} & 0^{1} & 0^{0} & 1^{1} & 0^{1} & 0^{1} & 0^{1} & 0^{1} & 0_2 \end{array}$$

Schreibweise: Summe$^{\text{Übertrag der vorherigen Stelle}}$

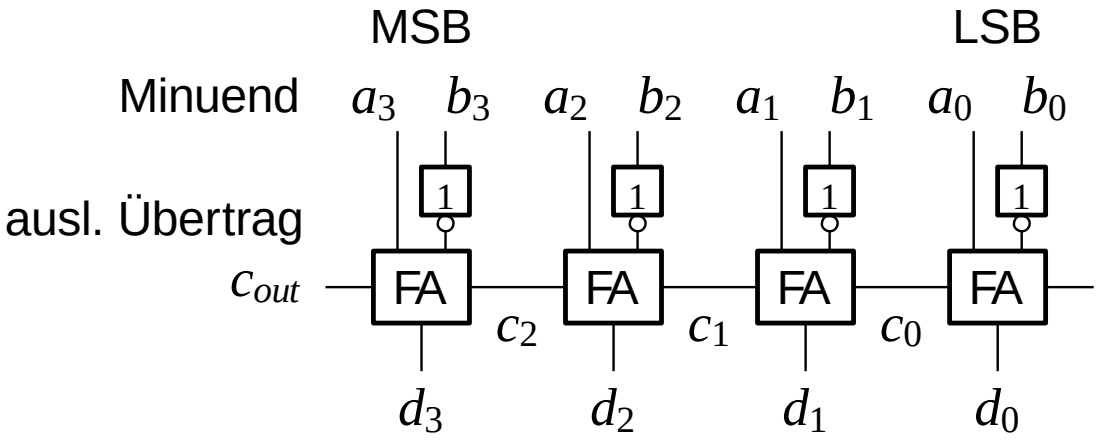
Rückführung der Subtraktion auf eine Addition mit dem B-Komplement des Subtrahenden bei fester Stellenzahl $n$: $d = a - b \quad \rightarrow \quad d = a +^{(2)} b$

Der dabei auftretende auslaufende Übertrag (carry out) $c_{n-1}$ wird ignoriert.

# Hardware-Realisierung der Subtraktion



**Beispiel: 4-Bit Subtraktion (Ripple-Carry-Adder mit 2-Komplement)**




$d = a - b$

$d = a +^{(2)} b$

$$d_\nu = a_\nu \oplus \overline{b}_\nu \oplus c_{\nu-1} \quad \text{für } \nu = 0,1,\ldots,n-1$$

$$c_\nu = a_\nu \wedge \overline{b}_\nu \vee a_\nu \wedge c_\nu \vee \overline{b}_\nu \wedge c_\nu$$

$$c_{-1} = 1 \quad \text{für 2-Komplement}$$

$$c_{out} = c_{n-1}$$

# Multiplikation (1) Beispiel



dezimal ($n = 2$)

$$
\begin{array}{llll}
1 & 5_{10} & \cdot & 1 \quad 3_{10} \\
\hline
 & & & 4^1 \; 5 \\
+ & & & 1^0 \; 5 \\
\hline
= & & & 1^0 \; 9^0 \; 5_{10}
\end{array}
$$

dual ($n = 4$)

$$
\begin{array}{llllllllll}
1 & 1 & 1 & 1_2 & \cdot & 1 & 1 & 0 & 1_2 & \\
\hline
 & & & & & 1 & 1 & 1 & 1 & \\
 & & & & 0 & 0 & 0 & 0 & & \\
 & & & 1 & 1 & 1 & 1 & & & \\
+ & & 1 & 1 & 1 & 1 & & & & \\
\hline
= & 1 & 1^{10} & 0^{10} & 0^{10} & 0^1 & 0^0 & 1^0 & 1_2 &
\end{array}
$$

Sukzessive stellenweise Multiplikation des Multiplikators ($0$ oder $1$) mit dem LSB beginnend bis hin zum MSB mit dem gesamten Multiplikanten und stellenrichtige Aufsummierung der Teilergebnisse (Linksverschiebung um $n-1$ Stellen):
$(a \cdot b = a \cdot b_0 \cdot 2^0 + a \cdot b_1 \cdot 2^1 + \ldots + a \cdot b_{n-1} \cdot 2^{n-1})$.

# Multiplikation (2)



$$
\begin{aligned}
a \cdot b &= p \\
b &= b_0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + \ldots + b_{n-1} \cdot 2^{n-1} \\
p &= a \cdot b_0 + a \cdot b_1 \cdot 2^1 + a \cdot b_2 \cdot 2^2 + \ldots + a \cdot b_{n-1} \cdot 2^{n-1} \\
b_v &\in \{0, 1\}
\end{aligned}
$$

Die Multiplikation kann so auf eine fortlaufende Addition und eine entsprechende Stellenverschiebung zurückgeführt werden.

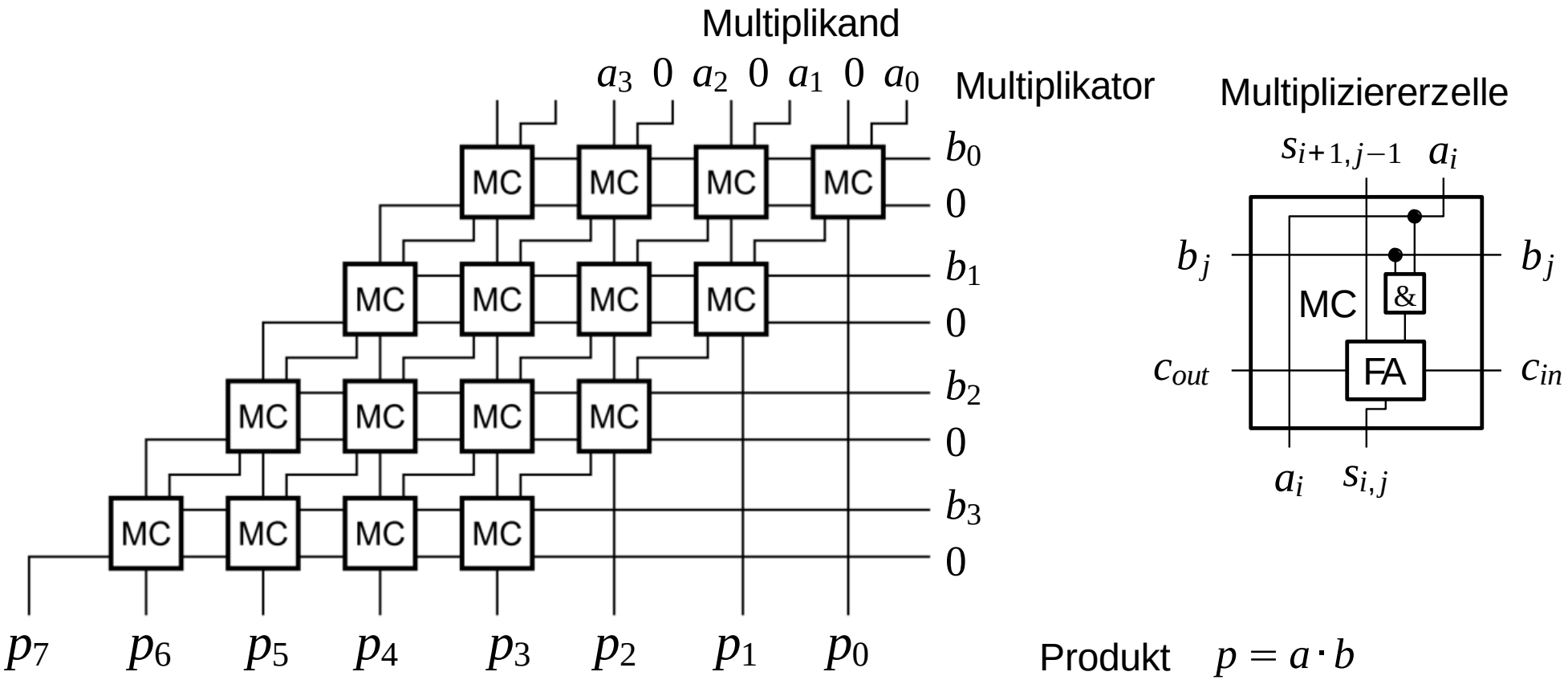
Eine Multiplikation mit $2$ bedeutet eine Verschiebung des Multiplikanten um eine Stelle nach links, wobei das LSB mit 0 aufgefüllt wird (analog für eine Multiplikation mit $2^k$ eine Verschiebung um $k$ Stellen).

$a \cdot 2^k \rightarrow (a_l = a_{l-k}$, für $l = n-1, \ldots, k)$ und $(a_l = 0$, für $l = k-1, \ldots, 0)$

# Hardware-Realisierung der Multiplikation



**Beispiel: 4-Bit Multiplikation (Ripple-Carry-Addition mit 1-Bit Multiplikation)**




Produkt $p = a \cdot b$

# Division Beispiel



dezimal ($n = 2$)

$$
\begin{array}{l}
\quad 59_{10} : 3_{10} = 19_{10} \\
-\ 3 \\
\hline
\quad 29 \\
-\ 27 \\
\hline
\quad\ \ 2 \\
\quad\ \underline{\ \ }
\end{array}
$$

dual ($n = 8$) ohne führende 0

$$
\begin{array}{lll}
\quad 1110\,11_2 : \ 11_2 = & & 1001\,1_2 \\
-\ 11 & \rightarrow 11 \leq 11 & \rightarrow 1\ \uparrow \\
\hline
\quad\ 0\,1\,0\,11 & \rightarrow 11 > 01 & \rightarrow 0\ \ \uparrow \\
\quad\ 11 & \rightarrow 11 > 010 & \rightarrow 0\ \ \ \ \uparrow \\
\quad\ \ 1\ 1 & \rightarrow 11 < 0101 & \rightarrow 1\ \ \ \ \ \ \uparrow \\
\quad\ -\ 1\ 1 & & \\
\hline
\quad\ \ 1\ 01 & & \\
\quad\ \ -\ 11 & \rightarrow 11 < 101 & \rightarrow 1\ \ \ \ \ \ \ \ \uparrow \\
\hline
\quad\ \ \ 10 & &
\end{array}
$$

# Division (2)



$$
\begin{aligned}
a : b &= q \\
q &= q_0 + q_1 \cdot 2^1 + q_2 \cdot 2^2 + \ldots + q_{n-1} \cdot 2^{n-1} \\
a &= b \cdot q \\
a &= b \cdot q_0 + b \cdot q_1 \cdot 2^1 + b \cdot q_2 \cdot 2^2 + \ldots + b \cdot q_{n-1} \\
&\quad \cdot 2^{n-1} \\
q_\nu &\in \{0, 1\}
\end{aligned}
$$

Bei der Division wird stellenweise mit dem MSB von $q$ beginnend das Produkt $b \cdot [q_\nu] \cdot 2^\nu$ von $a$ (durch Verschiebung) subtrahiert. Ist die Differenz positiv, so ist $q_\nu = 1$, sonst $0$. Mit der verbleibenden Differenz von $a$ wird analog weiter verfahren bis alle Stellen abgearbeitet sind. Als letzte Differenz bleibt der Divisionsrest übrig.

Die Division kann so auf eine fortlaufende Stellenverschiebung und eine entsprechende Subtraktion (Adddition mit 2-Komplement) zurückgeführt werden.

# Division (3)



Eine Division durch $2$ bedeutet eine Verschiebung des Dividenden um eine Stelle nach rechts, wobei das MSB mit 0 aufgefüllt wird (analog für eine Division durch $2^k$ eine Verschiebung um $k$ Stellen).

$a/2^k \rightarrow (a_{l-k} = a_l$, für $l = k, \ldots, n-1)$ und $(a_l = 0$, für $l = n-1, \ldots, n-k-1)$

Die Division ist in Hardware aufwendiger zu realisieren als die Multiplikation und benötigt daher allgemein auch mehr Rechenzeit bei der Abarbeitung.

Eine Alternative zu diesem Divisionsverfahren stellen verschiedene iterative Verfahren dar, die die Division ebenfalls auf Stellenverschiebung, Multiplikation und Addition zurückfüren.

# Division (4)



**Iterative Division mit dem Newton-Verfahren**

| | | | |
|---|---|---|---|
| $q$ | $=$ | $\frac{a}{b}$ | |
| $f(x)$ | $=$ | $\frac{1}{x} - b$ | Ansatzfunktion mit Nullstelle bei $x = \frac{1}{b}$ |
| $x_{i+1}$ | $=$ | $x_i - \frac{f(x_i)}{f^t(x_i)}$ | Iterationsvorschrift nach Newton |
| $f^t(x_i)$ | $=$ | $\frac{-1}{x^2}$ | Ableitung von $f(x)$ |
| $x_{i+1}$ | $=$ | $x_i \cdot (2 - x_i \cdot b)$ | Iterationsvorschrift, $x_{i+1}$ - verbesserte Näherung |
| $\frac{a}{b}$ | $=$ | $a \cdot x_n$ | für $n \to \infty$ |

$x_0$ - Startwert, $x_i$ - Näherungswert nach $i$ Iterationen

(Alternatives Verfahren: Iterative Division mit dem Goldschmidt-Algorithmus )

# Division (5)



**Beispiel:** $q = \frac{\bar{a}}{b} = \frac{\overline{15}}{7} = 15 \cdot \frac{\bar{1}}{7} = 2.142857142857\ldots$

Startwert: $x_0 = 0.1$

| $x_{i+1} =$ | $x_i \cdot (2 - x_i \cdot b)$ | Genauigkeit |
|---|---|---|
| $x_1 =$ | $0.1 \cdot (2 - 0.1 \cdot 7) = \underline{0.1}3$ | $10^{-1}$ |
| $x_2 =$ | $0.13 \cdot (2 - 0.13 \cdot 7) = \underline{0.14}17$ | $10^{-2}$ |
| $x_3 =$ | $0.1417 \cdot (2 - 0.1417 \cdot 7) = \underline{0.1428}4777$ | $10^{-4}$ |
| $x_4 =$ | $\ldots = \underline{0.14285714}2242$ | $10^{-8}$ |
| $x_5 =$ | $\ldots = \underline{0.142857142857}\ldots$ | $< 10^{-10}$ |

Lösung: $q = 15 \cdot x_5 = 2.142857142857\ldots$

# Fest/Gleit-kommazahlen
# Fixed/Floating-point numbers

# Zahlendarstellung im Computer, Datenformate



Die Zahlendarstellung im Computer (Datenformat) erfolgt in der Regel binär codiert in Formaten fester Länge $n$. Es wird das Dualzahlen-Stellenwertsystem verwendet. Es gelten die Rechenregeln für Dualzahlen.

Folgende Darstellungsvarianten werden allgemein unterschieden:

- vorzeichenlose natürliche Zahlen
- vorzeichenbehaftete ganze Zahlen
- Festkommazahlen (Festpunktzahlen)
- Gleitkommazahlen (Gleitpunktzahlen)

Durch die Darstellungsvariante selbst und die begrenzte Stellenzahl (Datenformate fester Länge $n$) sind die Gültigkeitsbereiche des Assoziativgesetzes und des Distributivgesetzes zu beachten.

# Gebrochene Zahlen im Stellenwertsystem



Die Zifferndarstellung einer allgemeinen gebrochenen Zahl $G_B$ lautet im Stellenwertsystem der Basis $B$ (Festkommadarstellung):

$$G_B = \pm(z_{n-1}\ldots z_1 z_0, z_{-1}z_{-2}\ldots z_{-m})_B = \pm((z_{n-1}\ldots z_1 z_0)_B + (0, z_{-1}z_{-2}\ldots z_{-m})_B)$$

Der Wert von $G_B$ bestimmt sich durch :

$$G_B = \pm \sum_{i=-m}^{n-1} z_i \cdot B^i = \sum_{i=-m}^{-1} z_i \cdot B^i + \sum_{i=0}^{n-1} z_i \cdot B^i$$

$$G_B = \pm((((z_{n-1} \cdot B + z_{n-2}) \cdot B \ldots) \cdot B + z_1) \cdot B + z_0 \quad \leftarrow \text{ganzzahlig}$$

$$+ \ ((z_{-m} \cdot B^{-1} + z_{-m+1}) \cdot B^{-1} \ldots) \cdot B^{-1} + z_{-1}) \cdot B^{-1}) \quad \leftarrow \text{echt gebrochen}$$

# Gebrochene Zahlen im Stellenwertsystem



Die Zifferndarstellung einer allgemeinen gebrochenen Zahl $G_B$ lautet im Stellenwertsystem der Basis $B$ (Gleitkommadarstellung):

$$G_B = \pm(0, z_{-1}z_{-2}\ldots z_{-m})_B \cdot B^{(\pm e_{k-1}\ldots e_1 e_0)_B} \quad \text{mit} \quad M_B = \sum_{i=1}^{m} z_{-i}B^{-i}, \quad E_B = \sum_{j=0}^{k-1} e_j B^j.$$

Der Wert von $G_B$ bestimmt sich durch :

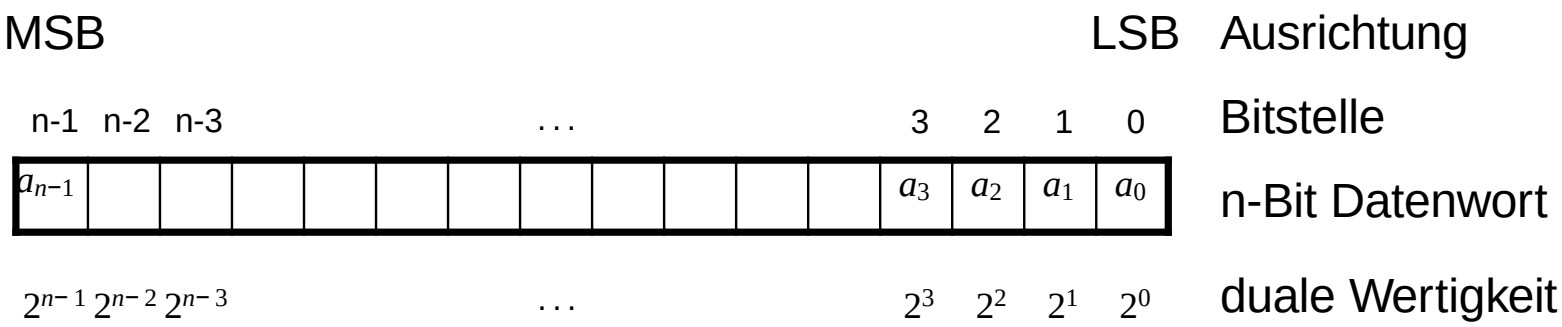
$$G_B = \pm \left(\sum_{i=1}^{m} z_{-i} \cdot B^{-i}\right) \cdot B^{\left(\pm \sum_{j=0}^{k-1} e_j B^j\right)} = \pm M_B \cdot B^{(\pm E_B)}$$

$M_B$ - ist eine echt gebrochene Zahl und wird als Mantisse bezeichnet. (Hidden bit?)
$E_B$ - ist eine ganze Zahl und wird als Exponent bezeichnet.

Der Exponent gibt an, um wieviele Stelle das Komma nach rechts (positiv) bzw. nach links (negativ) verschoben werden muss.

# N-Bit Datenformat (Dualzahlen-Stellenwertsystem)






Wert $Z$ bei Interpretation als reine Dualzahl:

$$Z = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \ldots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

Übliche Datenformate: 8 Bit, 16 Bit, 32 Bit, 64 Bit, . . .

Je nach Datenformat haben einzelne Bitstellen spezielle Funktionen, z.B. Vorzeichen, Exponent, . . .

Jede Zahlendarstellung hat daher ein gesondertes Datenformat.

# Festkommazahlen (fixed-point numbers)



Festkommazahlen werden zur Darstellung von gebrochenen Zahlen verwendet:

- Die Darstellung analog der Vorzeichen-Wert- oder 2-Komlement-Darstellung.
- Die Anzahl der Nachkommastellen wird vorher fest definiert ($f$-fracional part).
- Das Komma steht implizit immer an der selben Stelle (nicht abgespeichert).
- Die Zahl der Vorkommastellen ergibt sich aus der Differenz von Gesamtstellenzahl und Nachkommastellenzahl (Vorzeichen im MSB).

**n-stellige Festkommazahl mit m Nachkommastellen:**

$$z = d, f \quad \rightarrow \quad d_{n-1} d_{n-2} \ldots d_{n-m} f_{m-1} f_{m-2} \ldots f_0$$

kleinste Zahl für $|z| > 0$: $2^{-m}$ größte Zahl für $|z|$: $(2^{n-1} - 1) \cdot 2^{-m}$

**Anwendung**: Real-Arithmetik

# Konvertierung von Festkommazahlen (1)



**Beispiel:** Konvertierung einer Hexadezimalzahl $X_{16} = A45,8F$ (2 hexadezimale Nachkommastellen) in die wertgleiche Dezimalzahl $X_{10}$

$$X_{16} = A \quad 4 \quad 5, \quad 8 \quad F_{16}$$

$$X_{10} = A_{16} \cdot 16^2 + 4_{16} \cdot 16^1 + 5_{16} \cdot 16^0 + 8_{16} \cdot 16^{-1} + F_{16} \cdot 16^{-2}$$

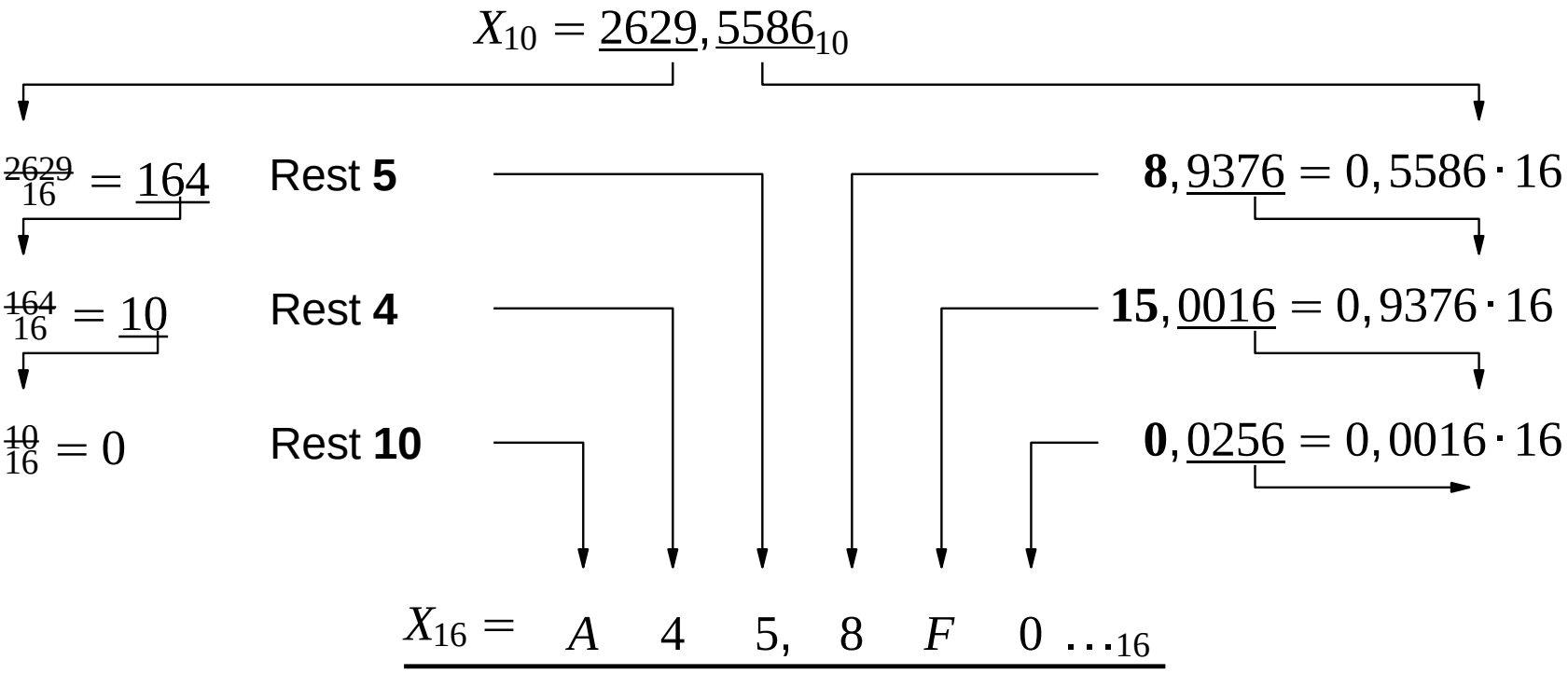
$$X_{10} = 10_{10} \cdot 16^2 + 4_{10} \cdot 16^1 + 5_{10} \cdot 16^0 + 8_{10} \cdot 16^{-1} + 15_{10} \cdot 16^{-2}$$

$$X_{10} = 2629,5586\ldots_{10}$$

# Konvertierung von Festkommazahlen (2)



**Beispiel:** Konvertierung einer Dezimalzahl $X_{10} = 2629,5586$ (4 dezimale Nachkommastellen) in die wertgleiche Hexadezimalzahl $X_{16}$



# Gleitkommazahlen (floating-point numbers)



Halblogarithmische Darstellung gebrochener Zahlen durch Mantisse und Exponent:

$$z = M \cdot 2^E$$

| | | |
|---|---|---|
| $V$ | - | Vorzeichen der Mantisse |
| $M$ | - | Mantisse (Signifikant $S$) |
| $E$ | - | Exponent (Charakteristik) |

**Vorteil:** wesentlich größerer Wertebereich und flexibler Nachkommabereich
**Nachteil:** geringere relative Genauigkeit

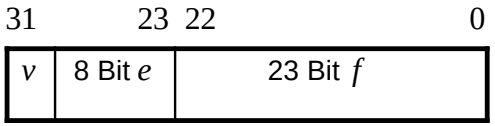
**Standard:** ANSI/IEEE 754 von 1985
(Institutes for Electrical and Electronic Engineers)

**Anwendung:** Real-Arithmetik

# Datenformate für Gleitkommazahlen



32-Bit Format (single)

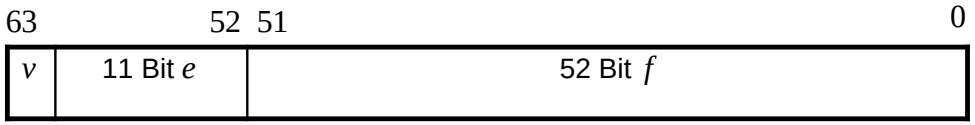



*v*-Vorzeichen

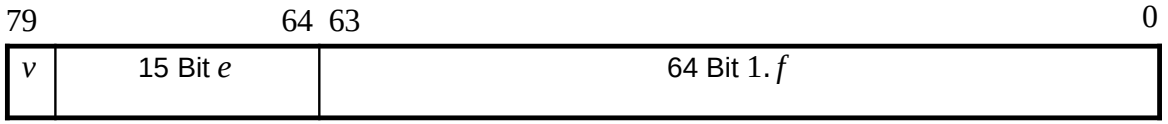
*e*-biased exponent

*f*-fractional part

64-Bit Format (double)




80-Bit Format (double-extended)



# Mantissendarstellung (1)



Darstellung der Mantisse $M$ als Festkommazahl (vorzeichenbehaftete gebrochene Zahl in Vorzeichen-Wert-Darstellung) mit Vorzeichen im MSB.

- **normalisierte Form:** Die führende $1$ und das Komma werden nicht explizit gespeichert, sondern nur der gebrochene Anteil $f$ (fractional part), (Ausnahme: double-extendet). → Vor der Verarbeitung ist die 1 zu ergänzen (entpacken). Das Komma steht immer rechts neben der 1.

$$M = (-1)^v \cdot 1, f$$

- **denormalisierte Form:** Die führende $1$ wird durch $0$ ersetzt. Dient zur Darstellung sehr kleiner Zahlen (Kennzeichnung im Exponenten) oder zur Exponentenanpassung bei Berechnungen. → geringere Genauigkeit als normalisierte Darstellung.

$$M = (-1)^v \cdot 0, f$$

# Mantissendarstellung (2)



**Wertebereich:** $1,0 \leq \quad |M| \quad < 2,0 \quad$ mit $\quad 0,0 \leq \quad f \quad < 1,0$

**Genauigkeit der Mantisse (binär - dezimal):**

| Datenformat | Binärstellen | max. rel. Genauigkeit | Dezimalstellen |
|---|---|---|---|
| single | 23 | $2^{-24} = 5,96 \ldots 10^{-8}$ | 7 |
| double | 52 | $2^{-53} = 1,11 \ldots 10^{-16}$ | 15 |
| double-extended | 64 | $2^{-64} = 5,42 \ldots 10^{-20}$ | 19 |

# Exponentendarstellung



Darstellung des Exponenten $E$ als vorzeichenbehaftete ganze Zahl in Basiswert-Darstellung) mit dem Basiswert $B_e$.

$$E = e - B_e \quad \text{bzw.} \quad e = E + B_e$$

**Wertebereich (ohne Ausnahmesituationen):**

| Datenformat | $b_e$ | $B_e$ | Wertebereich | | |
|---|---|---|---|---|---|
| single | 8 | 127 | -126 | $\leq E \leq$ | 127 |
| double | 11 | 1023 | -1022 | $\leq E \leq$ | 1023 |
| double-extended | 15 | 16383 | -16382 | $\leq E \leq$ | 16383 |

($b_e$ - Binärstellen (Exponent), $B_e$ - Basiswert (biased))

# Ausnahmesituationen der Gleitkommadarstellung (1)



Größter ($e = 2^{b_e} - 1$) und kleinster ($e = 0$) Exponentenwert sind für Ausnahmesituationen reserviert ($\rightarrow$ 4 Darstellungsvarianten):

| | Interpretation | $V$ | $e$ | $f$ | Zahlendarstellung |
|---|---|---|---|---|---|
| | Null | $v$ | 0 | 0 | $(-1)^v \cdot 0$ |
| 1. | denormalisiert | $v$ | 0 | $/= 0$ | $(-1)^v \cdot 0, f \cdot 2^{(1-B_e)}$ |
| 2. | normalisiert | $v$ | $0 < e < 2^{b_e} - 1$ | $/= 0$ | $(-1)^v \cdot 1, f \cdot 2^{(e-B_e)}$ |
| 3. | Unendlich | $v$ | $2^{b_e} - 1$ | 0 | $(-1)^v \cdot \infty$ |
| 4. | Not-a-Number | $v$ | $2^{b_e} - 1$ | $/= 0$ | NaN |

- zwei verschiedene Darstellungen der Null ($+0$ und $-0$).
- die Darstellung der Null erfolgt als denormalisierte Zahl.
- denormalisierte Zahlen werden mit $e = 1$ interpretiert.
- Bereichsüberschreitungen sind nicht möglich.
- Sonderdarstellungen für Unendlich, ($-\infty$ und $+\infty$).

# Ausnahmesituationen der Gleitkommadarstellung (2)



**Beispiele (single):**

| $v$ | $e$ | $f$ | Zahlenwert | Interpretation |
|---|---|---|---|---|
| 0 | 00000000 | 00 . . . 00 | $+0$ | pos. Null |
| 1 | 00000000 | 00 . . . 00 | $-0$ | neg. Null |
| 0 | 00000000 | 00 . . . 01 | $+0,00\ldots01 \cdot 2^{-126}$ | denormalisiert |
| 1 | 00000000 | 00 . . . 01 | $-0,00\ldots01 \cdot 2^{-126}$ | denormalisiert |
| 0 | 00000001 | 00 . . . 00 | $+1,00\ldots00 \cdot 2^{-126}$ | normalisiert |
| 1 | 00000001 | 00 . . . 00 | $-1,00\ldots00 \cdot 2^{-126}$ | normalisiert |
| 0 | 11111110 | 11 . . . 11 | $+1,11\ldots11 \cdot 2^{+127}$ | normalisiert |
| 1 | 11111110 | 11 . . . 11 | $-1,11\ldots11 \cdot 2^{+127}$ | normalisiert |
| 0 | 11111111 | 00 . . . 00 | $+\infty$ | pos. unendlich |
| 1 | 11111111 | 00 . . . 00 | $-\infty$ | neg. unendlich |
| 0 | 11111111 | 00 . . . 01 | NaN | Not-a-Number |

# Addition und Substraktion von Gleitkommadarstellung (1)



$$z_1 = M_1 \cdot 2^{E_1} = (-1)^{v_1} \cdot 1, f_1 \cdot 2^{(e_1 - B_e)}$$

$$z_2 = M_2 \cdot 2^{E_2} = (-1)^{v_2} \cdot 1, f_2 \cdot 2^{(e_2 - B_e)}$$

$$z_1 + z_2 = \left(M_1 + M_2 \cdot 2^{-(E_1 - E_2)}\right) \cdot 2^{E_1}$$

$$= \left((-1)^{v_1} \cdot 1, f_1 + (-1)^{v_2} \cdot 1, f_2 \cdot 2^{-(e_1 - e_2)}\right) \cdot 2^{(e_1 - B_e)}$$

Bei ungleichen Exponenten $e_1, e_2$ muß die Zahl mit dem kleineren Exponenten denormalisiert werden (Erhöhung des Exponenten bei gleichzeitiger Rechtsverschiebung des Signifikanten).

# Addition und Substraktion von Gleitkommadarstellung (2)



**Algorithmus für vorzeichenbehaftete Addition $z_s = z_1 + z_2$ :**

$$z_s = \left( M_1 + M_2 \cdot 2^{-(e_1 - e_2)} \right) \cdot 2^{E_1}$$

1. Wenn $e_1 < e_2$, dann vertausche $z_1$ und $z_2$.

2. Verschiebe $M_2$ um $e_1 - e_2$ Stellen nach rechts.

3. Addiere bzw. Subtrahiere $M_1$ und $M_2$ analog zu vorzeichenbehafteten ganzen Zahlen (Zweierkomplement). Bei negativem Ergebnis Konvertierung in die Vorzeichen-Wert-Darstellung.

4. Normalisierung und Rundung des Ergebnisses.

# Addition und Substraktion von Gleitkommadarstellung (3)



**Beispiel (single): ($^0$ - Zusatzstelle wegen 2-Komplement Rechnung)**

| $z$ | $v$ | $e$ | $s$ | Bemerkung | Dez. |
|---|---|---|---|---|---|
| $z_1$ | 0 | 1000 0001 | $^0 1,0000\ldots$ | normalisiert | $4,0$ |
| $z_2$ | 0 | 0111 1111 | $^0 1,1000\ldots$ | normalisiert | $1,5$ |
| $z_2$ | 0 | 1000 0001 | $^0 0,0110\ldots$ | denormalisiert | $1,5$ |
| $z_1 + z_2$ | 0 | 1000 0001 | $^0 1,0110\ldots$ | normalisiert | $5,5$ |
| $-z_2$ | 1 | 1000 0001 | $^1 1,1010\ldots$ | denormalisiert | $-1,5$ |
| | | | $\ldots$ | 2-Komplement | |
| $z_1 - z_2$ | 0 | 1000 0001 | $^0 0,1010\ldots$ | denormalisiert | $2,5$ |
| $z_1 - z_2$ | 0 | 1000 0000 | $^0 1,0100\ldots$ | normalisiert | $2,5$ |

# Multiplikation und Division von Gleitkommazahlen (1)



$$z_1 = M_1 \cdot 2^{E_1} = (-1)^{v_1} \cdot 1, f_1 \cdot 2^{(e_1 - B_e)}$$

$$z_2 = M_2 \cdot 2^{E_2} = (-1)^{v_2} \cdot 1, f_2 \cdot 2^{(e_2 - B_e)}$$

$$z_1 \cdot z_2 = M_1 \cdot M_2 \cdot 2^{(E_1 + E_2)}$$

$$= (-1)^{v_1} \cdot (-1)^{v_2} \cdot 1, f_1 \cdot 1, f_2 \cdot 2^{((e_1 + e_2 - B_e) - B_e)}$$

$$\frac{z_1}{z_2} = \frac{M_1}{M_2} \cdot 2^{(E_1 - E_2)}$$

$$= \frac{(-1)^{v_1}}{(-1)^{v_2}} \cdot \frac{1, f_1}{1, f_2} \cdot 2^{((e_1 - e_2 + B_e) - B_e)}$$

# Multiplikation und Division von Gleitkommazahlen (2)



**Algorithmus für $z_p = z_1 \cdot z_2$ bzw. $\frac{z_1}{z_2}$:**

$z_p = M_1 \cdot M_2 \cdot 2^{((e_1+e_2-B_e)-B_e)}$ bzw. $z_p = M_1/M_2 \cdot 2^{((e_1-e_2+B_e)-B_e)}$

1. Bestimmung des Vorzeichens($v_1 \neq v_2 \rightarrow$ negativ, $v_1 = v_2 \rightarrow$ positiv).
2. Multiplikation bzw. Division von $M_1$ und $M_2$ analog zu ganzen Zahlen.
3. Addition von $e_1$ und $e_2$ und Subtraktion von $B_e$ bzw. Subtraktion von $e_1$ und $e_2$ und Addition von $B_e$ analog zu vorzeichenbehafteten ganzen Zahlen.
4. Normalisierung und Rundung des Ergebnisses.

# Multiplikation und Division von Gleitkommazahlen (3)



**Beispiel (single):**

| $z$ | $v$ | $e$ | $s$ | Bemerkung | Dez. |
|---|---|---|---|---|---|
| $z_1$ | 0 | 1000 0001 | $1,0000\ldots$ | normalisiert | $4,0$ |
| $z_2$ | 0 | 0111 1111 | $1,1000\ldots$ | normalisiert | $1,5$ |
| $z_1 \cdot z_2$ | 0 | 1000 0001 | $1,1000\ldots$ | normalisiert | $6,0$ |
| $\frac{z_2}{z_1}$ | 0 | 0111 1101 | $1,1000\ldots$ | normalisiert | $0,375$ |

# Rundung von Gleitkommazahlen (1)



**Der IEEE 754 - Standard fordert:**

Das Ergebnis einer arithmetischen Operation soll das selbe sein, als würde exakt gerechnet und anschließend gerundet (Runden zum nächsten Wert, die Hälfte auf den geraden Wert (round-to-even)).

**Verschiedene Rundungsmodi:**

- nach $0$,
- nach $+\infty$ oder $-\infty$,
- zum nächsten Wert, bei $0,5$ auf den geraden Wert (round-to-even)

**Relevante Bitstellen für die Rundung (auf q Stellen, vom MSB gezählt):**

- guard Bit g - Bitstelle auf die gerundet wird (q Stelle vom MSB),
- round Bit r - Rundungsbit (q+1 Stelle vom MSB),
- sticky Bit s - Zusatzstellen (q+2, q+3, . . . Stellen vom MSB bis zum LSB)

# Rundung von Gleitkommazahlen (2)



**Beispiel (Rundung von $n = 8$-Bit Zahlen auf $q = 4$ Stellen):**

| Nr. | n-Bit Zahl | | | gerundet |
|---|---|---|---|---|
| 1 | 1,000 | 1000 | → | 1,000 |
| 2 | 1,000 | 1001 | → | 1,001 |
| 3 | 1,001 | 1000 | → | 1,010 |
| 4 | 1,001 | 1111 | → | 1,010 |
| 5 | 1,111 | 1000 | → | 10,00 |
| | $g$ | $r$ $-$ $s$ $-$ | | |

Ausschlaggebend für die Rundung ist, ob in den s-Bits (restlichen Stellen) überhaupt eine 1 steht, nicht die Anzahl. Für eine genaue Rundung sind mehrere s-Bits erforderlich (z.B. $s = 3$).

# Probleme beim Rechnen mit Gleitkommazahlen



- Berechnungen mit denormalisierten Zahlen ($e = 0$) (Ungenauigkeit),
- Überlauf (Wertebereichsüberschreitung),
- Unterlauf (begrenzte Darstellung betragsmässig kleiner Werte),
- Rundung.

**Assoziativ- und Distributivgesetze gelten nur eingeschränkt:**

Ursachen für die Differenzen sind die endliche Stellenzahl zur Zahlendarstellung und die Rundung der Ergebnisse.

$$a + (b + c) \neq (a + b) + c$$

$$a \cdot (b \cdot c) \neq (a \cdot b) \cdot c$$

$$(a + b) \cdot c \neq (a \cdot c) + (b \cdot c)$$

# Zusammenfassung



- Floating point representation common for large numbers
- Hardware requirements for floating point processing – especially Division – can be high
  - one needs to evaluate the trade-offs involved in having one