

Rechnen bei eingeschränkter Präzision

- Integer-Rechenwerke sind optimiert für die Verwendung ganzer Zahlen, nicht für das Rechnen mit Festkommazahlen !
- prinzipiell werden reelle Zahlen x aus einer Anwendung auf ganze n -Bit Zahlen x' abgebildet, die als Festkommazahlen interpretiert werden (z.B. durch Skalierung: $x' = \lfloor 2^\alpha \cdot x \rfloor$)
- drei Fälle:
 - 1) fester beschränkter Wertebereich von x : $x \in [a, b]$ mit $a < 0$
 \Rightarrow Zahl der Vorkommastellen: $s = \lceil \log_2(\max\{|a|, |b|\}) \rceil + 1$
 - 2) fester beschränkter Wertebereich von x : $x \in [a, b]$ mit $a \geq 0$
 \Rightarrow Zahl der Vorkommastellen: $s = \lceil \log_2 |b| \rceil$
 - 3) Wertebereich von x unbeschränkt, lediglich ein typischer Wert (z.B. Startwert) $x_0 \neq 0$ ist bekannt
 \Rightarrow Zahl der Vorkommastellen: $s = \lceil \log_2 |x_0| \rceil + \alpha$ mit α abgeschätztZahl der Nachkommastellen in allen drei Fällen: $q = n - s$

Rechnen bei eingeschränkter Präzision (Forts.)

- Probleme beim Rechnen mit Festkommazahlen:
 - Wahl eines guten Skalierungsfaktors 2^α , mit dem eine reelle Zahl x in eine Festkommazahl $x' = \lfloor x \cdot 2^\alpha \rfloor$ umgerechnet werden kann
 \Rightarrow Festkommazahl x' ist mit **Quantisierungsfehler** ϵ_x behaftet: $x' = x + \epsilon_x$
 - ist Zahl der Vorkommastellen s zu klein, so ist die **Dynamik** zu niedrig: Wahrscheinlichkeit für Überlauf ist groß !
 - ist Zahl der Nachkommastellen q zu klein, so ist die **Präzision** zu gering: Genauigkeit kann insbesondere für iterative Verfahren unzureichend sein !
 - der aus dem (im folgenden zugrunde liegenden) Zweierkomplement resultierende **asymmetrische Zahlenbereich** ist insbesondere bei kleinen Wortbreiten n oft ungünstig
 \Rightarrow Probleme z.B. bei Multiplikation oder Bildung des Absolutbetrags !
- betragsmäßig sehr kleine Festkommazahlen können mit einer negativen Vorkommastellenzahl s kodiert werden:
Beispiel: $n = 8, s = -4 \Rightarrow$ Kodierung von $x \in [2^{-12}, 2^{-4} - 2^{-12}]$ möglich !

Fehlerfortpflanzung

- Seien ε_a und ε_b die Fehler, mit denen zwei Festkommavariablen a' und b' behaftet sind: $a' = a + \varepsilon_a$, $b' = b + \varepsilon_b$
- für die Addition $a + b$ folgt:

$$a' + b' = a + \varepsilon_a + b + \varepsilon_b \Rightarrow \varepsilon_{a+b} = \varepsilon_a + \varepsilon_b$$
- für die Multiplikation $a \cdot b$ folgt:

$$a' \cdot b' = a \cdot b + a \cdot \varepsilon_b + b \cdot \varepsilon_a + \varepsilon_a \cdot \varepsilon_b + \varepsilon_{\text{mult}}$$

$$\Rightarrow \varepsilon_{a \cdot b} \gg a \cdot \varepsilon_b + b \cdot \varepsilon_a + \varepsilon_{\text{mult}}$$
 (wobei $\varepsilon_{\text{mult}}$ ein bei Multiplikation entstehender Fehler ist, z.B. durch Bildung eines n -Bit Wertes aus einem $2n$ -Bit Produkt)
- bei Anwendung einer Funktion $y = \phi(x)$ gilt:

$$y' = \phi(x + \varepsilon_x) + \varepsilon_\phi \approx \phi(x) + \varepsilon_x \cdot \phi'(x) + \varepsilon_\phi \Rightarrow \varepsilon_{\phi(x)} \approx \varepsilon_x \cdot \phi'(x) + \varepsilon_\phi$$
- bei einer Operation $y = \phi(x_1, \dots, x_k)$ gilt: $\hat{\Delta}_{\phi(x_1, \dots, x_k)} \approx \sum_{j=1}^k \hat{\Delta}_{x_j} \cdot \frac{\partial y}{\partial x_j} + \hat{\Delta}$

Truncating vs. Rounding

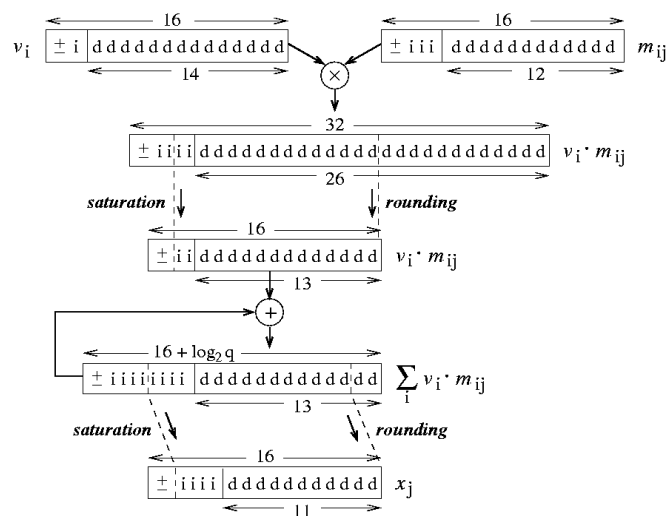
- beim Rechnen mit Festkommazahlen müssen oft von einer m -Bit Zahl x mit q Nachkommabitstellen die niedrigstwertigen r Bits abgeschnitten werden, um eine n -Bit Zahl mit $n < m$ zu erhalten
- im folgenden wird Gleichverteilung für x angenommen
- zwei Techniken:
 - 1) **Abschneiden („Truncating“)**:
 Abschneiden der Bitpositionen $x_{-q+r-1}, \dots, x_{-q+1}, x_{-q}$
 $\Rightarrow \varepsilon_x \in [-2^{-q+r} + 2^{-q}, 0]$, mittlerer Fehler: $E[\varepsilon_x] = -\frac{1}{2} \cdot (2^{-q+r} - 2^{-q})$
 - 2) **Runden („Rounding“)**:
 Abschneiden der Bitpositionen $x_{-q+r-1}, \dots, x_{-q+1}, x_{-q}$ und Addition von 2^{-q+r} , falls $(x_{-q+r-1} \dots x_{-q+1} x_{-q})_2 \cdot 2^{-q} \geq 2^{-q+r-1}$, d.h. falls gilt: $x_{-q+r-1} = 1$
 $\Rightarrow \varepsilon_x \in [-2^{-q+r-1} + 2^{-q}, 2^{-q+r-1}]$, mittlerer Fehler: $E[\varepsilon_x] = -\frac{1}{2} \cdot 2^{-q}$
- Runden ist stets vorzuziehen, wird bei Festkomma-Arithmetik i.a. aber nicht durch Hardware unterstützt !

Sättigung

- Eine andere Möglichkeit, aus einer m -Bit Festkommazahl eine n -Bit Zahl (mit $n < m$) zu erhalten, besteht im Abschneiden von r führenden *Vorkommabitstellen*
- Abschneiden der Bitpositionen $x_{s-1}, x_{s-2}, \dots, x_{s-r}$ ist fehlerfrei möglich, wenn gilt: $x_{s-1} = x_{s-2} = \dots = x_{s-r} = x_{s-r-1}$
 $\Rightarrow \epsilon_x \in [-2^{s-1}, 2^{s-1}]$, d.h. der resultierende Fehler liegt in der gleichen Größenordnung wie die Zahl x !
- alternativ ist eine **Sättigung** („*Saturation*“) denkbar, wird von heutiger Integer Arithmetik-Hardware i.a. aber nicht unterstützt:
 Wenn eine der abgeschnittenen Bitpositionen x_{s-2}, \dots, x_{s-r} ungleich dem Vorzeichen x_{s-1} ist, so wird in $x_{s-r-1} \dots x_0$ der größtmögliche darstellbare positive oder negative Wert generiert
 $\Rightarrow \epsilon_x \in [-2^{s-1} + 2^{s-r-1}, 2^{s-1} - 2^{s-r-1}]$
- typische Anwendung: Addition zweier n -Bit Zahlen

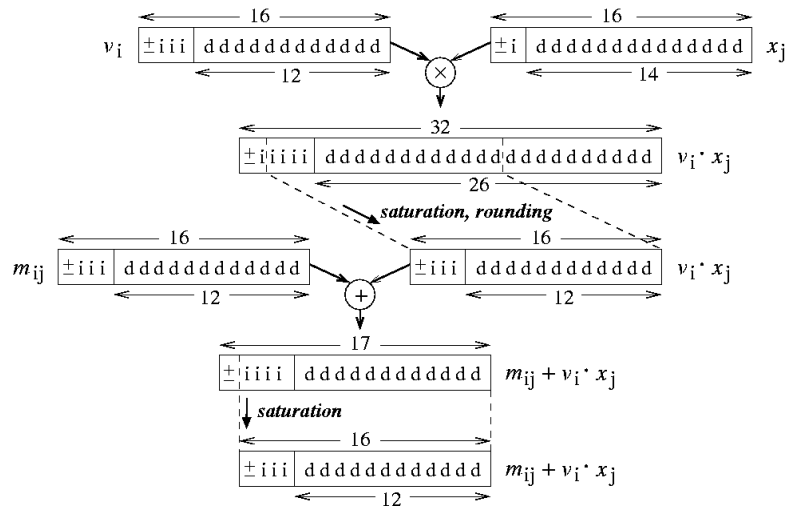
Beispiel 1:

Berechnung von $x_j = \sum_{i=1}^q v_i \cdot m_{ij}$ mit 16-Bit Festkommazahlen:



Beispiel 2:

Berechnung von $m_{ij} = m_{ij} + v_i \cdot x_j$ mit 16-Bit Festkommazahlen :



Beispiel 3:

Berechnung von $m_{ij} = m_{ij} + \eta \cdot (v_i - m_{ij}) \cdot x_j$:

