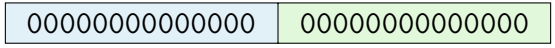
Fixkommazahlen

- Aufteilung der Bits in Vor- und Nachkommastellen
- Beispiel: 16.16 Fixkommazahl:



- Gewicht der Bits ab Komma: 2^{-1}, 2^{-2}, ...

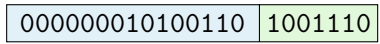
Fixkommazahlen: Beispiel

00000000001101	11010000000000
Vorkommast. (16)	Nachkommast. (16)

- Vorkommastellen: $1101_2 = 13_{10}$
- Nachkommastellen: $2^{-1} + 2^{-2} + 2^{-4} = 0.5 + 0.25 + 0.0625 = 0.8125$

$\Rightarrow 1101110100000000000_2$ als 16.16 Fixkommazahl: 13.8125_{10}

Quiz: Zahlen zuordnen 1

Welchen Wert hat 

?

- ☐ 166.609375
- ☐ 165.069375
- ☐ 38.87566

Fixkommazahlen: Arithmetik

Addition/Subtraktion:

- Komma muss an gleicher Stelle sein
- ansonsten keine weiteren Operationen notwendig

Multiplikation/Division:

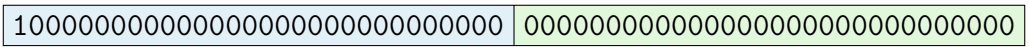
- Beim Multiplizieren zweier Festkommazahlen addieren sich die Kommapositionen

$$\boxed{110}\boxed{100} \times \boxed{1110}\boxed{11} = \boxed{1011111}\boxed{11100}$$

- Shifts notwendig, um das Ergebnis wieder in das richtige Format zu bringen
- Division: Multiplikation mit Kehrwert

Warum Fließkommazahlen?

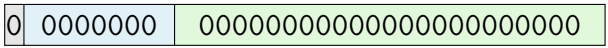
- Problem: Anzahl der Vor- und Nachkommastellen fix
  - Was, wenn man sehr große und sehr kleine Zahlen speichern möchte?
  - Beispiel: 2^{29} und 2^{-30}:



Vorkommastellen (30) Nachkommastellen (30)

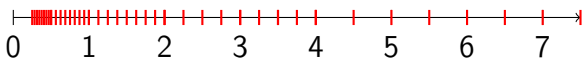
- teils unnötiger Speicherverbrauch

Fließkommazahlen - Aufbau



$$SignBit \times Mantisse \times 2^{Exponent-Bias} \tag{1}$$

- Feste Anzahl an signifikanten Stellen
- Größerer Wertebereich als Fixkommazahlen
- Konsequenz: Höhere Genauigkeit bei kleinen Zahlen

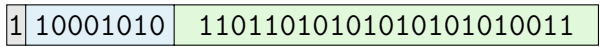


Aufbau

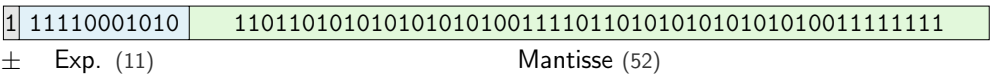
- Normalisiert: $1 \leq \textit{Mantisse} < 2$
- Führende Eins nicht abgespeichert
- Exponent(gespeichert) = Exponent(real) + Bias
  - Bias statt Zweierkomplement
  - Lexikografischer Vergleich statt Subtraktion und Vergleich mit 0
  - In der Theorie weniger Operationen

Datentypen `float`/`double`

	Größe	Dezimalziffern	Abs. Min.	Abs. Max.
`float` (Single Prec.)	32 Bit	≈ 7	$\approx 1.18 \cdot 10^{-38}$	$\approx 3.4 \cdot 10^{38}$



`double` (Double Prec.)	64 Bit	≈ 15	$\approx 10^{-308}$	$\approx 10^{308}$



Fließkommazahlen - Beispiel

Konvertierung von 37.75 zu einer Fließkommazahl:

1. Konvertierung zu einer Fixkommazahl:

100101	11

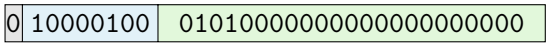
2. Komma so weit verschieben, sodass nur noch eine 1 vor dem Komma steht:

1	0010111

3. *Nachkommastellen* = *Mantisse*; $\#$*Rechtsshifts* + *Bias* = *Exponent*:

0	10000100	00101110000000000000000
$\pm$	Exp. (8)	Mantisse (23)

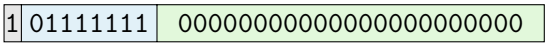
Quiz: Zahlen zuordnen 2

Welchen Wert hat 

?

- ☐ $1.3125_{10} \times 2^{132}$
- ☐ 42.0_{10}
- ☐ $1.3125_{10} \times 2^{5}$

Quiz: Zahlen zuordnen 3

Welchen Wert hat 

?

- ☐ -1.0_{10}
- ☐ -127.0_{10}
- ☐ 0

Addition und Subtraktion

- Kleineren Wert auf selben Exponenten bringen wie großen Wert (denormalisieren)
- Mantissen addieren bzw. subtrahieren
- Mantisse entsprechend der Genauigkeit runden
- Ergebnis normalisieren

Subtraktion – Beispiel

+	2^1	1.0000	–	+	2^0	1.5000

Gleicher Exponent

+	2^1	1.0000	–	+	2^1	0.7500

Mantrisse subtrahieren

$=$

+	2^1	0.2500

Normalisieren

$=$

+	2^{-1}	1.0000

Multiplikation und Division

- Exponenten addieren bzw. subtrahieren
- Mantissen multiplizieren bzw. dividieren (Führende 1 beachten)
- Mantisse entsprechend der Genauigkeit runden
- Ergebnis normalisieren

Probleme bei Genauigkeit

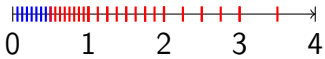
- Rundung: Ergebnis muss wieder in FP-Darstellung gespeichert werden
  - Verschiedene Rundungsmodi, Standard: *round to nearest, ties to even*
- Absorption: Addition/Sub. von sehr großer und sehr kleiner Zahl
  - Keine Veränderung der großen Zahl wg. Rundung
  - Beispiel: $1000000.00f + 0.01f = 1000000.00f$
- Auslöschung: Subtraktion großer ähnlicher Zahlen
  - Subtraktion verstärkt Rundungsfehler
  - Beispiel: $1000000.1f - 1000000.0f = 0.125f \neq .1f$
  - Grund: $1000000.1f$ tatsächlich dargestellt als 1000000.125

Assoziativität und Distributivität

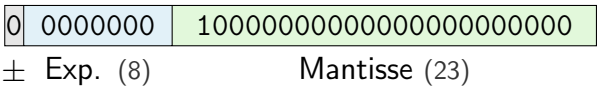
- Nicht assoziativ
  - $(x + y) + z \neq x + (y + z)$
  - $(x \times y) \times z \neq x \times (y \times z)$
- Nicht distributiv
  - $x(y + z) \neq (xy) + (xz)$
- Achtung: `-ffast-math` (`-Ofast`) in GCC ignoriert diese zwecks Geschwindigkeit
- Weiterführend: What Every Computer Scientist Should Know About Floating-Point Arithmetic[1]

[1] `https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf`

Denormale Zahlen / Subnormale Zahlen

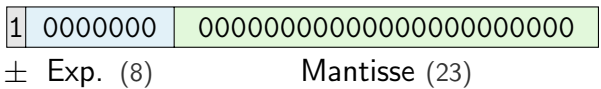


- Zahlen deren Exponent kleiner ist, als eine normalisierte Darstellung zulassen würde
- Beispiel, single precision, normalisiert: $1.0_2 \times 2^{-127}$
- Denormalisiert: $0.1_2 \times 2^{-126}$
- Exponent hat speziellen Wert: alle Bits 0



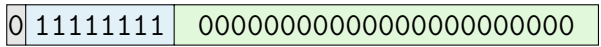
Null mit Vorzeichen

- Null: Exponent und Mantisse alle Bits 0
- Sign-Bit kann gesetzt sein $\rightarrow +/-0$ möglich



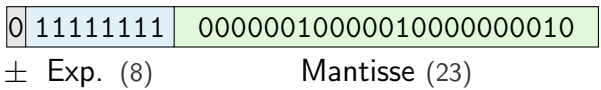
Unendlich / Infinity / ∞

- Alle Bits in Exponent $= 1$
- Alle Bits in Mantisse $= 0$
- je nach Sign-Bit: $+/-$ Unendlich
- z.B. Ergebnis bei $x/0$



Not a Number / NaN

- Alle Bits in Exponent $= 1$
- Mantisse $\neq 0$
- z.B. Ergebnis bei $0/0$ und $\infty - \infty$
- Jede Operation mit NaN ergibt i.d.R. wieder NaN
- Jeder Vergleich mit NaN `false` (Ausnahme: $\neq$)



Quiz: Sonderfälle 1

Welches Ergebnis hat NaN == NaN?

- ☐ true
- ☐ false
- ☐ Segmentation Fault

Quiz: Sonderfälle 2

Welches Ergebnis hat NaN != Infinity?

- ☐ true
- ☐ false
- ☐ 1

Quiz: Sonderfälle 3

Welches Ergebnis hat -Infinity < Infinity?

- ☐ true
- ☐ false
- ☐ Arithmetic Exception: Invalid Operation

Quiz: Sonderfälle 4

Welches Ergebnis hat 10 != NaN?

- ☐ true
- ☐ false
- ☐ Infinity

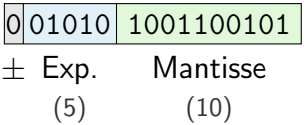
Quiz: Sonderfälle 5

Welches Ergebnis hat 5.0 / 0.0?

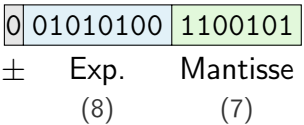
- [ ] -Infinity
- [ ] NaN
- [ ] Infinity
- [ ] Arithmetic exception: Division by Zero

Weitere Floating Point Formate

- 16 Bit half precision / half



- Brain Floating Point / bfloat



- Extended Formate

Zusammenfassung

Fließkommazahlen:

- Großer Wertebereich bei weniger Speicherplatzverbrauch
- Ungenauigkeiten durch Rundung (Absorption, Auslöschung)
- Arithmetik nicht trivial

Fixkommazahlen:

- Schnelle Arithmetik
- Gleichbleibende Genauigkeit im gesamten Wertebereich
- Kommaposition fest $\rightarrow$ Hoher Speicherplatzverbrauch bei großen Wertebereichen