

Vorlesung 9

- Kommazahlen in Binär
 - Einführung
 - Exponentialdarstellung
 - Gleitkommadarstellung
 - Normierung
 - Umsetzung im Rechner
 - Beispiel
 - Schritt 1
 - Schritt 2
 - Schritt 3
 - Sonderwerte für den Exponent
 - Berechnungsfehler wegen Gleitkommadarstellung
- Formatierung der Ausgabe (***iomanip***)

Kommazahlen in Binär

Eine Kommazahl in Binär

- Eine Kommazahl ist öfters in Dualsystem nicht vollständig darstellbar bzw. speicherbar

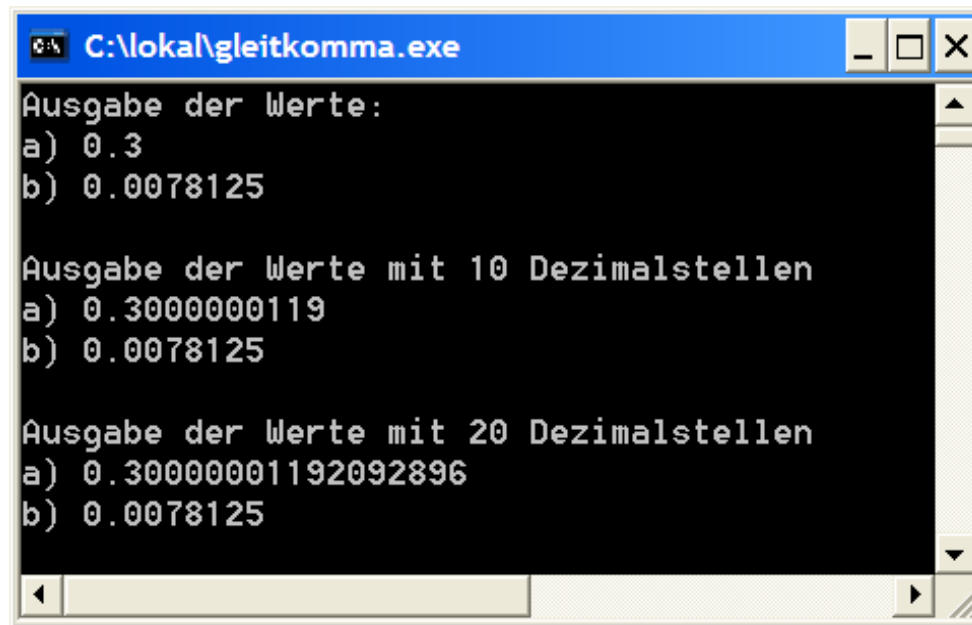
- Beispiel: eine Potenz von 2 ist immer genau speicherbar.

$$2^{-7} = 0.0078125_{10} = 0.0000001_2$$

Aber die Zahl $0,3 = 0,010011001100110011..$

$$= 0,01001$$

Sie kann wegen der Periode nicht genau gespeichert werden. Irgendwo muss die Reihe von Nullen und Einen abgeschnitten werden.



```
C:\lokal\gleitkomma.exe
Ausgabe der Werte:
a) 0.3
b) 0.0078125

Ausgabe der Werte mit 10 Dezimalstellen
a) 0.3000000119
b) 0.0078125

Ausgabe der Werte mit 20 Dezimalstellen
a) 0.30000001192092896
b) 0.0078125
```

Auf dem Bild sieht man durch die „cout“ Ausgabe von weiteren Kommastellen, wie die Zahl 0,3 nicht genau als 0,3000000000000000 gespeichert ist.

-
- Die Kommazahlen werden im Computer als binäre Gleitkommazahlen (auch genannt: Fließkommazahlen) gespeichert.
D.h. 0.3 wird im Computer nicht genau als 0,01001100110011.... sondern ein bisschen anders.
 - Als erster Schritt wird die Exponentialdarstellung von Zahlen eingeführt. Aus dieser Darstellung her kann man die Gleitkommadarstellung sehr simpel bestimmen.

Exponentialdarstellung von Zahlen

Diese Art Darstellung ist sehr oft in Ingenieurwissenschaft in üblichen Taschenrechnern benutzt und ist das Fundament zur Gleitkommadarstellung

$$z = \pm m p^{\pm e}$$

Mit m=Mantisse

e=Exponent

P=Basis des Exponenten

Beispiel: Mit Exponentenbasis = p = 10
die Zahl 16,35 kann geschrieben werden als $0,1635 \times 10^2$ bzw.
 $1,635 \times 10^1$ usw.

oder

0,0237 kann als $2,37 \times 10^{-2}$ oder $0,237 \times 10^{-1}$ usw.

Gleitkommadarstellung nach Norm

Allgemeine Form:

$$z = (-1)^s \times 1.f \times 2^{e-o}$$

Mit:

- **S** = *Vorzeichen*, „Signum“, Sign(0=Positiv, 1=Negativ)
- **f** = *Fraktion*, „fraction“.
- **e** = *Exponente*.
- **o** = Offset Halber Wertbereich des Exponenten(Wg.+/-)

Die **Mantisse** ist in die normalisierte Form **1.f**

d.h. vorderste Dualziffer ungleich 0 in Vorkommaposition bringen(die 1 vor dem Komma wird als „hidden bit“ bezeichnet und in der Regel nicht mit abgespeichert).

Umsetzung der Normierung

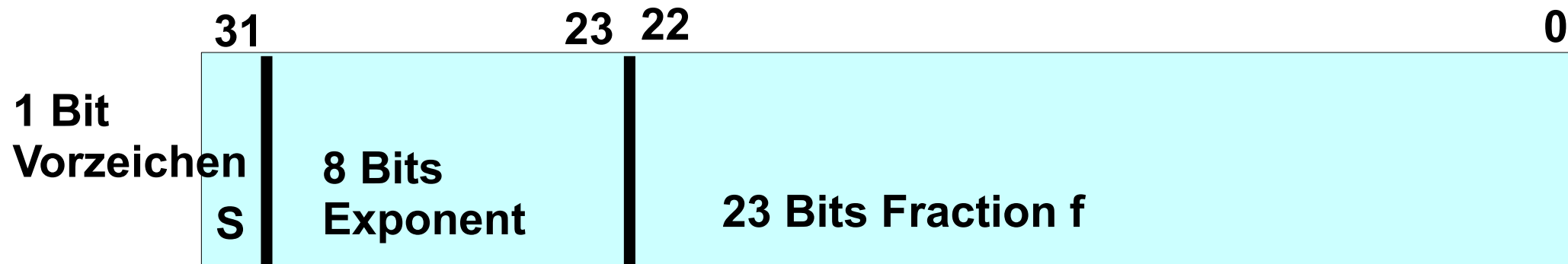
Gleitkommadarstellung im Rechner

Standards sind für einfache und doppelte Genauigkeit verfügbar:

- **32 Bit (Single Precision = Einfache Genauigkeit)**
- **64 Bit (Double Precision = Doppelgenauigkeit).**

Single Precision von Gleitkommazahlen

32 Bits Format

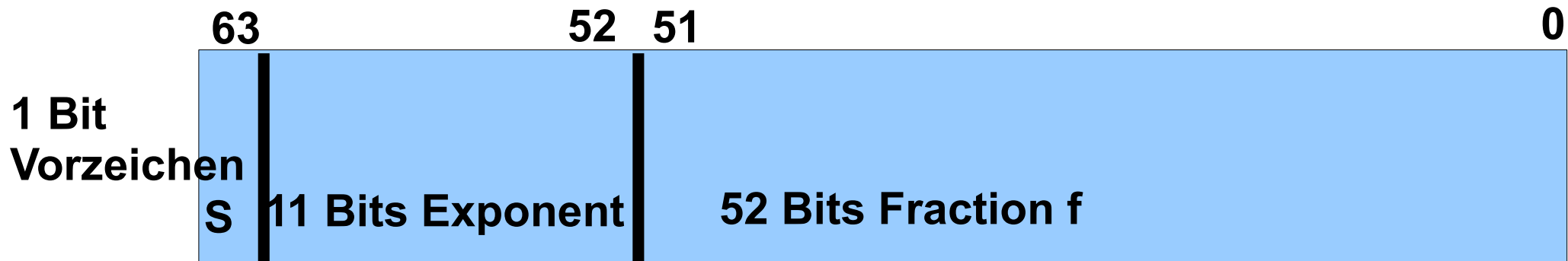


Namen für den Datentyp dieses Formats in verschiedenen Programmiersprachen

- C/C++/java: *float*
- Visual Basic: *single*
- Fortran/PASCAL: *real*

Double Precision von Gleitkommazahlen

64 Bits Format



Namen für den Datentyp dieses Formats in verschiedenen Programmiersprachen

- C/C++/java: *double*
- Visual Basic: *double*
- PASCAL: *double*
- Fortran: *double precision*

Beispiel für Darstellung in Gleitkommazahlen

nur für Single Precision

Schritt 1

45.625 in Dezimalsystem

Der ganze und Bruchteil werden separat umgewandelt.
Für die Umwandlung des Bruchteils allein siehe nächste Folie

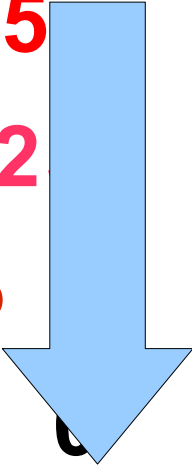
$$\left. \begin{array}{l} 45 \rightarrow 101101_2 \\ .625 \rightarrow .101_2 \end{array} \right\} 101101.101_2$$

Auf dieser Art und Weise erhalten wir die **binäre Darstellung** aller Kommazahlen, die **noch nicht** in die genormte Gleitkommadarstellung ist.

Umwandlung des Bruchteils in binär

Multiplikationsverfahren

Beispiel mit 0.8125 in Binär

$$\begin{array}{lcl} 0.8125 & * 2 = & \mathbf{1}.625 \\ 0.\mathbf{625} & * 2 = & \mathbf{1}.2 \\ 0.\mathbf{25} & * 2 = & \mathbf{0}.5 \\ 0.\mathbf{5} & * 2 = & \mathbf{1}.0 \end{array}$$


$$0.8125_{10} \rightarrow 0.\mathbf{1101}_2$$

Beispiel für Darstellung in Gleitkommazahlen

Schritt 2 *nur für Single Precision*

$$45.625 \rightarrow 101101.101_2$$

Nun muss die binäre Zahl in die Exponentialdarstellung.
Dies bedeutet einfach die Komma zu verschieben ohne den Wert zu verändern

$$101101.101_2 \rightarrow 1.01101101 \times 2^5 \quad (5 \text{ Stelle "geschoben"})$$

Jetzt sind Vorzeichen **s**, Fraktion **f** und Exponent **e** direkt erhaltbar.

$$s = 0 \text{ (positiv)}$$

$$f = 01101101$$

$$o = 127 \text{ (immer so für die einfache Genauigkeit)}$$

$$e = 127 + 5 = 132 = 10000100$$

Beispiel für Darstellung von Gleitkommazahlen

Schritt 3 *nur für Single Precision*

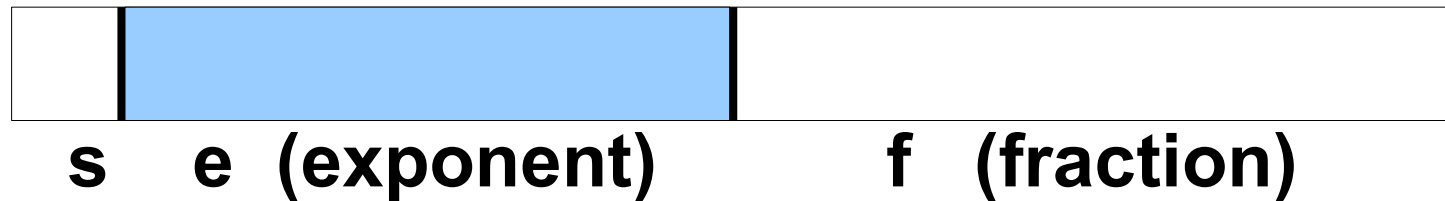
Mit solchen binären Werten für das Vorzeichen (s), die Fraktion (f) und das Exponent (e) wird die Gleitkommadarstellung der binäre Zahl folgendes

Ergebnis:

| | | |
|----------|-----------------|--|
| 0 | 10000100 | 011011010000...00 |
| s | exp. | Fraction, auf 23 Bit aufgefüllt |

Gleitkommazahlen

Sonderwerte für den Exponent



Sonderwerte:

$e = 0$ (00000000_2) ist der kleinste Exponent

$e = 255$ (11111111_2) ist für **Unendlich** reserviert, **alle** Ziffern der Mantisse sind 0

$e = 255$ (11111111_2) und mindestens eine Ziffer der Mantisse ungleich 0 ist **Not a Number (NaN)** d.h. diese Gleitkommazahl entspricht keine Zahl, ist aus einer fehlerhaften Berechnung erhalten worden.

Fehler wegen Gleitkommadarstellung

- Zahlen werden approximativ dargestellt
z.B. 0,3 ist nicht genau als binäre Gleitkommazahl darstellbar

Darstellungsungenauigkeit.cpp

- Die signifikanten Stellen (6 für single precision) erzeugen Fehler bei Operationen.

z.B. $8.0 + 0,0000001 = 8,000000000 \neq 8,0000001 !!!$

Der Wert 0,0000001 kann nicht mit Werten höher als 2,0 addiert werden.

SignifikativeStelle.cpp

- Die Ungenauigkeit werden vergrößert durch iterativen Verfahren. Die iterativen Operationen (Schleifen) vergrößern die Darstellungsfehler.

Ungenauigkeitsschleife.cpp

Formatierung der Ausgabe

iomanip

- In dieser Vorlesung haben wir unsere Ausgabe formatiert.
- Um die Kommastellen auszugeben muss man *setprecision* (*Anzahl von Kommastellen*) einsetzen

Zum Beispiel: `cout << setprecision(10) << 0.3;`

- Diese „Manipulator“ sowie andere Nützlichen befinden sich in der C++ Bibliothek *iomanip*

Siehe Beispielprogramm in der Datei iomanip.pdf