Float **Vs.** Double auf ARM

Betrachten wir zunächst den folgenden Codeausschnitt:

Input:

```
float x  = 3.141592653589793238
double z = 3.141592653589793238
printf("x=%f\n", x)
printf("z=%f\n", z)
printf("x=%20.18f\n", x)
printf("z=%20.18f\n", z)
```

Output:

```
x=3.141593
z=3.141593
x=3.141592741012573242
z=3.141592653589793116
```

Floating point numbers in C verwenden die **IEEE 754-Encoding**.

Diese Art der Encoding verwendet ein **Vorzeichen**, einen **Signifikanten** und einen **Exponenten**.

Aufgrund dieser Encoding werden viele Zahlen geringfügig geändert, damit sie gespeichert werden können.

Außerdem kann sich die Anzahl der signifikanten Stellen geringfügig ändern, da es sich um eine *binäre* Darstellung und nicht um eine *Dezimalzahl* handelt.

Bei einfacher Genauigkeit (Float) erhalten Sie *23 Bit* Signifikanz, *8 Bit* Exponent und *1* Vorzeichenbit.

Auf der anderen Seite verwendet die doppelte Genauigkeit (Double) *52 Bit* Signifikanz, *11 Bit* Exponent und *1* Vorzeichenbit, was der doppelten Anzahl von Bits entspricht, die zur Darstellung eines Floats verwendet werden (32 Bits).

Doubles haben immer 53 signifikante Bits und Floats haben immer 24 signifikante Bits (mit Ausnahme von Denormalen, Unendlichkeiten und NaN-Werten, die jedoch Gegenstand einer anderen Frage sind).

Hierbei handelt es sich um Binärformate, und Sie können nur deutlich über die Genauigkeit ihrer Darstellung in Form von Binärziffern (Bits) sprechen.

Der Grund, warum es als Double bezeichnet wird, ist, dass die Anzahl der verwendeten Bytes zum Speichern doppelt so groß ist als die eines Floats. (Dies fasst jedoch sowohl den Exponenten als auch den Signifikanten um).

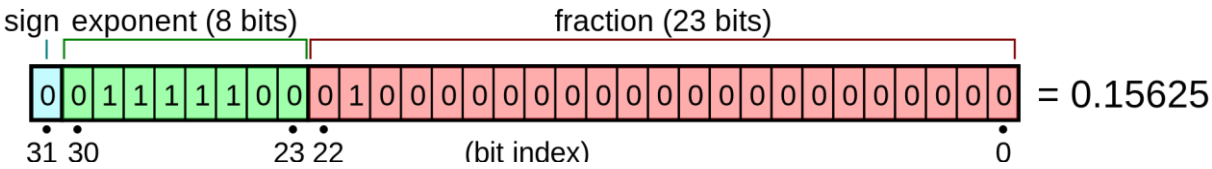
Der IEEE 754-Standard (von den meisten Compilern verwendet) weist dem Signifikanten relativ mehr Bits zu als dem Exponenten (23 bis 9 für Float gegenüber 52 bis 12 für Double), weshalb die Genauigkeit mehr als verdoppelt wird.

Beispielweise von Float: (Wikipedia)

Die wahre Bedeutung enthält 23 fraction-bits rechts vom Binärpunkt und ein implizites führendes Bit (links vom Binärpunkt) mit dem Wert 1, sofern der Exponent nicht mit allen Nullen gespeichert ist.

Somit erscheinen im Speicherformat nur 23 fraction-bits des Signifikanten, aber die Gesamtgenauigkeit (total-precision) beträgt jedoch 24 Bit (log10 (224) ≈ 7,225 Dezimalstellen).

Die Bits sind wie folgt angeordnet:



Der reale Wert, der von bestimmten 32-Bit-Daten (binary32) angenommen wird (mit einem zeichen, Exponent e (die 8-Bit-Ganzzahl ohne Vorzeichen), 23-bit fraction), ist:

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\dots b_{23})_2 - 127} \times (1.b_{22}b_{21}\dots b_0)_2,$$

was ergibt:

$$\text{value} = (-1)^{\text{sign}} \times 2^{(e-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right).$$

In diesem Beispiel:

- $\text{sign} = b_{31} = 0,$
- $(-1)^{\text{sign}} = (-1)^0 = +1 \in \{-1, +1\},$
- $e = b_{30}b_{29}\dots b_{23} = \sum_{i=0}^{7} b_{23+i}2^{+i} = 124 \in \{1, \dots, (2^8 - 1) - 1\} = \{1, \dots, 254\},$
- $2^{(e-127)} = 2^{124-127} = 2^{-3} \in \{2^{-126}, \dots, 2^{127}\},$
- $1.b_{22}b_{21}\dots b_0 = 1 + \sum_{i=1}^{23} b_{23-i}2^{-i} = 1 + 1 \cdot 2^{-2} = 1.25 \in \{1, 1 + 2^{-23}, \dots, 2 - 2^{-23}\} \subset [1; 2 - 2^{-23}] \subset [1; 2).$

somit:

- $\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625.$

Hinweis:

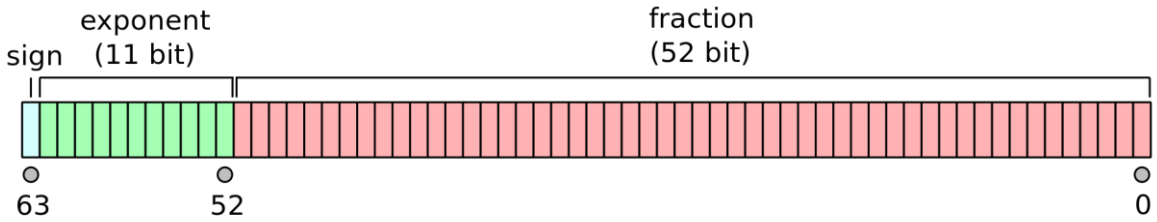
- $1 + 2^{-23} \approx 1.000\,000\,119,$
- $2 - 2^{-23} \approx 1.999\,999\,881,$
- $2^{-126} \approx 1.175\,494\,35 \times 10^{-38},$
- $2^{+127} \approx 1.701\,411\,83 \times 10^{+38}.$

Beispiel von Double: (Wikipedia)

Das Format wird mit dem Hochkomma geschrieben, das ein implizites Integer-Bit mit dem Wert 1 enthält.

Wenn die 52 Bits der Fraktion (F) im Speicherformat angezeigt werden, beträgt die Gesamtgenauigkeit daher 53 Bits (ungefähr 16 Dezimalstellen, 53 log10 (2) ≈ 15.955).

Die Bits sind wie folgt angeordnet:



Der reale Wert, der von einem gegebenen 64-Bit- Grundlage mit doppelter Genauigkeit, mit einem gegebenen voreingenommenen Exponenten ***e*** und einem 52-Bit-Fraction angenommen wird, ist:

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\dots b_0)_2 \times 2^{e-1023}$$

Oder:

$$(-1)^{\text{sign}}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023}$$

Zwischen 2^{52} = 4,503,599,627,370,496 und 2^{53} = 9,007,199,254,740,992 sind die darstellbaren Zahlen genau die Integers.

Für den nächsten Bereich von 253 bis 254 wird alles mit 2 multipliziert, sodass die darstellbaren Zahlen gerade sind usw. Für den vorherigen Bereich von 251 bis 252 beträgt der Abstand 0,5 usw.

Der Abstand als fractionteil der Zahlen im Bereich von 2n bis 2^{n+1} beträgt 2^{n-52}. Der maximale relative Rundungsfehler beim Runden einer Zahl auf die nächste darstellbare Zahl beträgt daher 2^{-53}.

Die 11-Bit-Breite des Exponenten ermöglicht die Darstellung von Zahlen zwischen 10^{-308} und 10^{308} mit voller Genauigkeit von 15-17 Dezimalstellen. Die subnormale Darstellung erlaubt durch Abstriche bei der Präzision noch kleinere Werte bis ca. 5 × 10−324.

Die subnormale Darstellung erlaubt durch compromising precision noch kleinere Werte bis ca. 5×10^{-324}.

Zur Implementierung

Doppelte Genauigkeit ist in den Normen nicht erforderlich (mit Ausnahme des optionalen Anhangs F von C99, der die IEEE 754-Arithmetik abdeckt), aber bei den meisten Systemen entspricht der doppelte Typ doppelter Genauigkeit.

Auf 32-Bit-x86-Systemen mit erweiterter Genauigkeit entsprechen einige Compiler jedoch möglicherweise **nicht** dem C-Standard und / oder die Arithmetik weist möglicherweise eine doppelte Rundung auf.

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

https://en.wikipedia.org/wiki/Double-precision_floating-point_format#Double-precision_examples