Floating Point Zahlen

→ für was stehen die einzelnen Stellen?

↳ immer für Potenzen der jeweiligen Basis

e.g. dezimal (Basis 10) 157.9

$1 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0 + 9 \cdot 10^{-1}$

exp. der Basis pro Stelle:

3	2	1	0	.	-1	-2	-3
10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}

ausserdem: $1.579 \cdot 10^2 = 157.9$ also $\cdot 10^{exp}$ heisst "punkt" verschieben

$157.9 \cdot 10^{-2} = 1.579$

→ genau so mit Basis 2

exponent	3	2	1	0	-1	-2	-3
Zweierpotenz	2^3	2^2	2^1	2^0	2^1	2^{-2}	2^{-3}
Wert (B=10)	8	4	2	1	0.5	0.25	0.125

↳ mit $\cdot 2^{exp}$ multiplizieren heisst auch "Punkt" verschieben wenn wir in binär reden

$1.101 \cdot 2^2 = 110.1$

→ umrechnen

→ aufteilen: $1 + 0.9$ → ganz-zahliger Teil normal als ob Zahl

→ Nachkomma-Stellen umrechnen

immer die Zahl links vom Punkt (0 oder 1)

immer · 2 rechnen

⇒	0.9	– 0 →	0.9	→ 1.8
	1.8	– 1 →	0.8	→ 1.6
repeating part	1.6	– 1 →	0.6	→ 1.2
	1.2	– 1 →	0.2	→ 0.4
	0.4	– 0 →	0.4	→ 0.8
	0.8	– 0 →	0.8	→ 1.6
	1.6	– 1 →	0.6	→ 1.2
	1.2	– 1 →	0.2	→ 0.4
	0.4	– 0 →	0.4	→ 0.8
	0.8	– 0 →	0.8	→ 1.6
	1.6	– 1 →	0.6	

1 1 0 0 ⇒ resultat

immer in die nächste Zeile schreiben

Welcher Teil ist periodisch, also wiederholt sich? (wenn überhaupt)

1.0

$0.1\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\ldots$

$1.\overline{1\,1\,1\,0\,0}$

Normalized floating-point system

Basis: $b_i \in \{0, \dots, b-1\}$ → mit Basis $b=2$ $b_i \in \{0,1\}$

$b_0 \neq 0$

$F(b, p, e_{min}, e_{max})$

$e \in \{e_{min}, \dots, e_{max}\}$

Präzision:

(inkl. leading 1)

d.h. ihr könnt p Stellen benutzen:

p stellen

$b_0 . b_1 b_2 b_3 \dots b_{p-1} \cdot b^e$

"normalized" heisst es ist immer 1.... und nie 0....

e.g. $0.25_{10} = 0.01_2$

$0.01_2 \rightarrow 1 \cdot 2^{-2}$

→ Zwei floating-point Zahlen addieren.

1) Beide Zahlen auf den gleichen Exponenten bringen
2) Zahlen addieren (Punkt und 2^e beim Resultat nicht vergessen)
3) Re-normalisieren
4) Runden
   - ↳ abrunden falls nächste Stelle 0 ist

auf 4 Stellen: $[1.001]010$

- ↳ aufrunden falls nächste Stelle 1 ist

$[1.001]+1\,010$

d.h. jetzt diese 1 "aufrunden" d.h. diese 1 wird zu 10 → 1 als carry

1.010

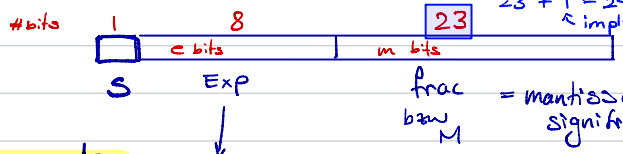
⇒ Woher kommt das? (Ihr müsst floats umrechnen etc. können, diese Seite ist einfach für die, die es interessiert)

single precision: float

$$\rightarrow (-1)^S \cdot M \cdot 2^E$$

$F^*(2, \boxed{24}, -126, 127)$

stored as:



Exponent

$E = Exp - Bias$ — $Bias = 2^{e-1} - 1$

(→ a way of encoding E to be ⊖ and ⊕, different from two's complement → this way is better for comparison)

→ single p. → Exp is the binary number stored in those 8 bits ⇒ Exp is in range $\{0, \dots, 255\}$ as we have 8 bits ⇒ 0 to 255. Exp = 0 → denorms, Exp = 255 → special vals

→ so to get the E from your Exp bits, given $Exp \in \{1, \dots, 254\}$ do:

$E = Exp - Bias$, Bias for float $= 2^{8-1} - 1 = 2^7 - 1 = 128 - 1 = 127$

⇒ range:

$e_{min} = 1 - 127 = -126$

$e_{max} = 254 - 127 = 127$

⇒ i.e. to get our number $(-1)^2 \cdot M \cdot 2^E$ we use $E = Exp - Bias$ but E is encoded in memory through Exp.

mantissa / significand

⇒ encoded with an **implicit leading 1** → "normalized"

$M = 1.\underbrace{xxxx \dots x}_{23 \text{ digits}}$ → these are the bits that are actually stored.

min: all zeros (0000…0) ≙ $M = 1.0$

max: all ones ≙ $M = 1.1 \dots 1$

Denorms: Exp is all zeros

→ $E = -Bias + 1 = -126$

→ frac all zeros → ± 0

→ frac ≠ 0 → small values closer to 0 "subnorms"

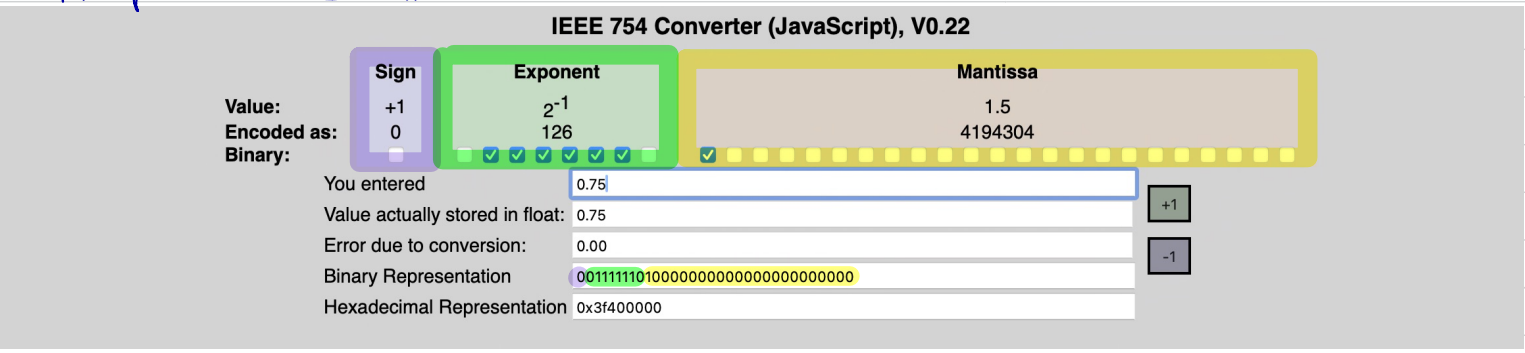
Special Vals: Exp all ones

↳ frac all zero → ± inf

↳ frac ≠ 0 → NaN

Bsp. aus https://www.h-schmidt.net/FloatConverter/IEEE754.html

Hier sehen wir, wie die Zahl tatsächlich gespeichert wird. Wenn wir dem Schema der vorherigen Seite folgen, sehen wir wie die bits interpretiert werden.



→ sign bit ist 0 ⇒ keine negative Zahl
(tatsächlich wissen wir auch Zahl ≠ 0 da 0 eine denormalisierte Zahl ist d.h. das Exponentenfeld ist nur 0...0)

→ Die bits im Exp feld sind 01111110
⇒ wir können E ausrechnen via

$$E = Exp - Bias = 01111110 - Bias$$
$$= 126_{10} - 127_{10} = -1_{10}$$

→ Die Mantissa ist 10...0

Jetzt können wir unsere Zahl zusammensetzen:

$$+1.1 \cdot 2^{-1} = 0.11_2 = 0.75_2$$

2^{-1}: weil wir mit Basis $b = 2$ rechnen

1.1: weil wir eine normalized Zahl betrachten (Exp weder nur 0 noch nur 1) haben wir eine implizite 1

$F^*(2, 4, -2, 2)$ — Basis 2

$p = 4$

$E \in \{e_{min}, e_{max}\} = \{-2, 2\}$

Welche Zahlen sich darstellen?

State the following numbers in F*(2, 4, –2, 2):
1. the largest number;
2. the smallest number;
3. the smallest non-negative number.

Compute how many numbers are in the set F*(2, 4, –2, 2).

! heisst normalisiert
⇒ die Zahlen müssen also immer 1.xxx sein

wir können in unserem System mit $p = 4$ genau 3 bits variieren.
(das 4te bit ist ja schon durch die leading 1 gefüllt)
⇒ müssen bei jeder Frage überlegen wie wir die 3 bits und den exponent wählen

1) grösste Zahl? → frac feld so gross wie mögl. → 111
E so gross wie mögl. → 2
} $1.111 \cdot 2^2$

2) kleinste Zahl? einfach ⊖ die grösste Zahl d.h. Sign bit $s = -1$

3) Zahl > 0, "am nächsten bei 0" → E am kleinsten wählen
→ die 3 bits in frac so klein wie mögl.

⇒ 000
-2
} $1.000 \cdot 2^{-2}$

Nicht vergessen, es muss ja immer norm. sein.

→ Wie viele Zahlen können dargestellt werden?

$80 \rightarrow 5(2 \cdot 2^3)$

pro E:
- 2^3: □□□ → 3 bits, jeweils zwei Mögl. (entweder 0, 1)
- 2: immer ⊕ und ⊖
- 5: die mögl. E sind $-2, -1, 0, 1, 2$ i.e. 5