Technische Universität München
Institut für Informatik
Prof. Dr. Thomas Huckle
Michael Rippl
Fabio Gratl

SoSe 2018

Numerisches Programmieren, Übungen

Musterlösung 1. Übungsblatt: Zahlendarstellung, Rundungsfehler

1) Umrechnung von Zahlen

a) Zahldarstellung in allgemeinem System: $\sum_{i=0}^{N} r_i \text{Basis}^i$ mit $r_i \in \{0, 1, \ldots, \text{Basis} - 1\}$

	dezimal	binär	trinär	hexadezimal
Basis	10	2	3	16
darzust. Zahl	19	10011	201	13
darzust. Zahl	47	101111	1202	2f
darzust. Zahl	511	111111111	200221	1ff

b) Schriftliches Dividieren der Brüche im binären System analog zum dezimalen:

$-\frac{1}{7} = -1_2 : 111_2$ (binär) :

$$
\begin{array}{llllllllll}
- & 1 & 0 & 0 & 0 & : & 1 & 1 & 1 & = \quad -0.00\overline{100} \quad \text{bzw. } -0.\overline{001} \\
-) & & 1 & 1 & 1 & & & & & \\
\hline
 & & & & 1 & 0 & 0 & 0 & &
\end{array}
$$

$\frac{1}{10} = 1_2 : 1010_2$ (binär) :

$$
\begin{array}{llllllllll}
 & 1 & 0 & 0 & 0 & 0 & : & 1 & 0 & 1 & 0 \quad = \quad 0.000\overline{1100} \quad \text{bzw. } 0.0\overline{0011} \\
-) & & 1 & 0 & 1 & 0 & & & & & \\
\hline
 & & & 1 & 1 & 0 & 0 & & & & \\
-) & & & 1 & 0 & 1 & 0 & & & & \\
\hline
 & & & & & 1 & 0 & 0 & 0 & 0 &
\end{array}
$$

2) Binärdarstellung von ganzen Zahlen

Beachte: n Bytes $= 8n$ Bits!

Wert 1. Bit	-2^{8n-1}
Wert 2. Bit	2^{8n-2}
kleinste Zahl x_{min}	-2^{8n-1}
größte Zahl x_{max}	$2^{8n-1} - 1$

Byte	Bits	x_{min}	x_{max}
1	8	-2^7	$2^7 - 1$
2	16	-2^{15}	$2^{15} - 1$
3	24	-2^{23}	$2^{23} - 1$
4	32	-2^{31}	$2^{31} - 1$
n	8n	-2^{8n-1}	$2^{8n-1} - 1$

Der Vorteil des Zweierkomplement gegenüber der Vorzeichenbit-Darstellung, ist die eindeutige Darstellbarkeit der 0 im Zweierkomplement. In der Version mit Vorzeichenbit gibt es sowohl +0 als auch -0.

3) Assoziativgesetz

dezimal	binär
-8	-1000_2
11	1011_2
0.75	0.11_2

Damit gilt:

$$\begin{aligned}(-8+11)+0.75 &= (-1000_2+1011_2)+0.11_2 = (11_2)+0.11_2\\ &= 11.11_2 = \boxed{3.75} \text{ (kein Runden nötig)}\\ -8+(11+0.75) &= -1000_2+(1011_2+0.11_2) = -1000_2+(1011.11_2) \overset{Runden!}{=} -1000_2+(1100_2)\\ &= 100_2 = \boxed{4}\end{aligned}$$

Das Ergebnis ist offensichtlich abhängig von der Reihenfolge der abgearbeiteten Operationen. Diese Rundungsfehler können auch zu sehr großen Fehlern führen. Dabei sei auf die Aufgabe zur Stabiliätsanalyse auf dem nächsten Blatt verwiesen.

4) Gleitkomma-Zahlen

a) Schriftliches Dividieren (wie in Aufg. 1): $-\frac{11}{10} = -1011_2 : 1010_2$ (binär) :

$$\begin{array}{llllllllll}
 & 1 & 0 & 1 & 1 & : & 1 & 0 & 1 & 0 \\
-) & 1 & 0 & 1 & 0 & & & & & \\
\hline
 & & & & 1 & 0 & 0 & 0 & 0 & \\
-) & & & & & 1 & 0 & 1 & 0 & \\
\hline
 & & & & & & 1 & 1 & 0 & 0 \\
-) & & & & & & 1 & 0 & 1 & 0 \\
\hline
 & & & & & & & & 1 & 0
\end{array} \quad 0 \quad 0 \quad 0$$

$= -1.000\overline{1100}$ bzw. $-1.0\overline{0011}$

Damit erhält man insgesamt: $-1.0\overline{0011} \cdot 2^0$

Alternative:

$\frac{11}{10}$	$\cdot$	1	$=$	$\frac{11}{10}$	≥ 1	1
						.
$\frac{1}{10}$	$\cdot$	2	$=$	$\frac{2}{10}$	< 1	0
$\frac{2}{10}$	$\cdot$	2	$=$	$\frac{4}{10}$	< 1	0
$\frac{4}{10}$	$\cdot$	2	$=$	$\frac{8}{10}$	< 1	0
$\frac{8}{10}$	$\cdot$	2	$=$	$\frac{16}{10}$	≥ 1	1
$\frac{6}{10}$	$\cdot$	2	$=$	$\frac{12}{10}$	≥ 1	1
$\frac{2}{10}$	$\cdot$	2	$=$	$\frac{4}{10}$	< 1	0
$\frac{4}{10}$	$\cdot$	2	$=$	$\frac{8}{10}$	< 1	0
$\frac{8}{10}$	$\cdot$	2	$=$	$\frac{16}{10}$	≥ 1	1
$\frac{6}{10}$	$\cdot$	2	$=$	$\frac{12}{10}$	≥ 1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

b) Rundungsregel (korrektes Runden) für eine Zahl $x = (-1)^{\nu} \cdot 2^{e} \cdot 1.x_1 x_2 \ldots x_{t-1} | x_t x_{t+1} x_{t+2} \ldots$:

Nr.	Fallbed.	Rundungsvorschrift
Einfacher Fall: Abrunden		
1)	$x_t = 0$	$rd(x) = (-1)^{\nu} \cdot 2^{e} \cdot 1.x_1 x_2 \ldots x_{t-1}$
Einfacher Fall: Aufrunden		
2)	$x_t = 1 \wedge$ $\lvert x_t x_{t+1} x_{t+2} \ldots \neq \lvert 1000000000 \ldots$	$rd(x) = (-1)^{\nu} \cdot 2^{e} \cdot (1.x_1 x_2 \ldots x_{t-1} + 2^{-(t-1)})$
Runden zur näheren geraden Mantissen-Zahl:		
3)	$x_{t-1} \lvert x_t x_{t+1} \ldots = 0 \lvert 100000000 \ldots$	$rd(x) = (-1)^{\nu} \cdot 2^{e} \cdot 1.x_1 x_2 \ldots x_{t-1}$
4)	$x_{t-1} \lvert x_t x_{t+1} \ldots = 1 \lvert 100000000 \ldots$	$rd(x) = (-1)^{\nu} \cdot 2^{e} \cdot (1.x_1 x_2 \ldots x_{t-1} + 2^{-(t-1)})$

Erläuterungen zu den verschiedenen Fällen:
Fall 1) und 2) stellen den normalen Fall dar, dass eine reelle Zahl nicht genau in der Mitte zwischen den beiden nächsten Maschinenzahlen liegt; es wird zum näheren Nachbarn gerundet. Fall 3) und 4) bewirken ein Runden mit Mantissenende $x_{t-1} = 0$ für reelle Zahlen, die genau zwischen zwei Maschinenzahlen liegen. In Fall 2) und 4) wird mit $+2^{-(t-1)}$ das letzte Bit um eins erhöht (und eventuell ein Übertrag durchgeführt).

Frage: Wie kann eine Floating Point Einheit auf der CPU unendlich viele Stellen zum Runden ausrechnen (z. B. bei periodischen Nachkommastellen)?
Das ist nicht nötig. Z. B. muss bei der Division lediglich bekannt sein, ob ein x_{t+1} un-

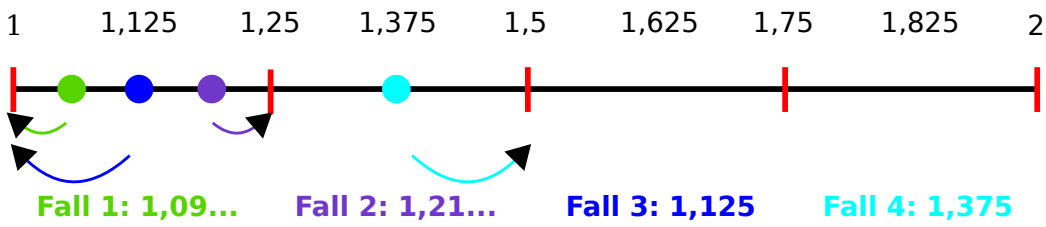


Abbildung 1: Veranschaulichung von Rundungen auf dem Zahlenstrahl

gleich 0 ist. Diese Information ist über den Rest erhältlich, der ab dem Berechnen von der Stelle x_{t+1} übrig bleibt.

Beispiel:
Es stehen zwei Mantissenbits zur Verfügung, also: $1, x_1 x_{t-1} | x_t x_{t+1} x_{t-2} ...$
Damit sind die folgenden Zahlen exakt darstellbar:

$$\begin{aligned} 1,00_2 &= 1_{10} \\ 1,01_2 &= 1,25_{10} \\ 1,10_2 &= 1,5_{10} \\ 1,11_2 &= 1,75_{10} \end{aligned}$$

Beispiele für alle vier Fälle aus der obigen Tabelle:

Nr.	Binärzahl	Dezimalzahl	ab-/aufrunden
1)	$1,\underline{00}\|011_2$	$1,09375_{10}$	abrunden
2)	$1,00\|111_2$	$1,21875_{10}$	aufrunden
3)	$1,0\underline{0}\|100_2$	$1,125_{10}$	abrunden
4)	$1,0\underline{1}\|100_2$	$1,375_{10}$	aufrunden

Abbildung 1 veranschaulicht die vier Fälle auf dem Zahlenstrahl.
Für unsere Zahl $-\frac{11}{10} = -1.0\overline{0011} \cdot 2^0$ aus Teilaufgabe i) bedeutet das:

Bit-Verwendungszweck	gesetztes Bit für $-1.0\overline{0011} \cdot 2^0$
Vorzeichen (1 Bit) ('-' wird zu '1')	1
Exponent (8 Bits) (0 + 127 = 127)	0 1 1 1 1 1 1 1
Mantisse (23 Bits)	0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1

Das letzte Bit der Mantisse ergibt sich nach der Rundungsregel (s.o.).

c) Die Zahl $x = 1 + 2^{-30}$ ist nicht exakt darstellbar (zuwenig Stellen).
Die Zahl wird als rd(x) = 1 abgespeichert (vgl. Rundungsregeln).

Absoluter Fehler: $f_{abs} := |x - rd(x)| = 2^{-30}$

Relativer Fehler: $f_{rel} := |f_{abs}/x| < 2^{-30}$

d) Definition: Maschinengenauigkeit = Die größte positive Zahl ε_{Ma}, so dass $1 \oplus \varepsilon_{Ma} = 1$.

Die kleinste darstellbare Zahl größer als 1 ist $1 + 2^{-23}$ (23 Bits echt für Mantissen-Nachkommastellen frei, da Normierung '1.' nicht extra gespeichert werden muss!). Folglich liegt ε in einer Größenordnung von 2^{-24}.

e) Lösung: siehe Aufgabe 7 bei den Beobachtungen.

5) Fehlerabschätzung von Zeitschrittweiten

Floating Point Darstellung von 1/60:

Wir starten mit einfacher Division:

	1	0	0	0	0	0	0	:	1	1	1	1	0	0	= $0.000001\overline{0001}$ bzw. $0.00\overline{0001}$
-)		1	1	1	1	0	0								
					1	0	0	0	0	0	0				
-)						1	1	1	1	0	0				
									1	0	0				

Für die Mantisse stehen 23 Bit zur Verfügung. Die erste '1' an der 6-ten Nachkommastelle wird allerdings nicht explizit abgespeichert. D. h. erst ab der 30-ten Nachkommastelle können die Ziffern nicht mehr abgespeichert werden:

	0,000001	0001	0001	0001	0001	0001	000[1
Nr. der Nachkommastelle:	7	11	15	19	23	27	

Mit der Normalisierung und dem nicht-Abspeichern der führenden '1' kann die Zahl deshalb bis auf die 29. Stelle genau abgespeichert werden.

Wie wird die Zahl dann als floating-point Zahl abgespeichert?
Sign: 0 für positive Zahlen
Exponent: -6+127 = 0b1111001
Mantisse: 0b00010001000100010001001 mit nach oben gerundeter letzten Stelle!

0	01111001	00010001000100010001001
Sign	Exponent	Mantissa

Floating point number: $(1) \cdot 2^{-6-23} \cdot 8947849 = 0.016666667535901069641113 2...$

Kleines Beispielprogramm für Zweifler:

```
#include <iostream>

int main()
{
        float l = (1.0/60.0);
        std::cout << l << std::endl;
        std::cout << (void*)*(unsigned long*)&l << std::endl;
}
/*
Output:
0.0166667
0x3c888889
*/
```

Wie kann man das Problem umgehen?

- Positionsänderung relativ zur Startposition berechnen.

- Als Zeitschrittweite eine 2er-Potenz wählen (z.B. 1/64).
- Rechengenauigkeit erhöhen.
- Fehler ignorieren, wenn die Simulationsdauer klein genug ist.

6) Ermittlung von π nach Archimedes

a) Quadrat:

$$\begin{aligned} s &= \sqrt{2} \\ U &= 4\sqrt{2} \approx 5.6568 \end{aligned}$$

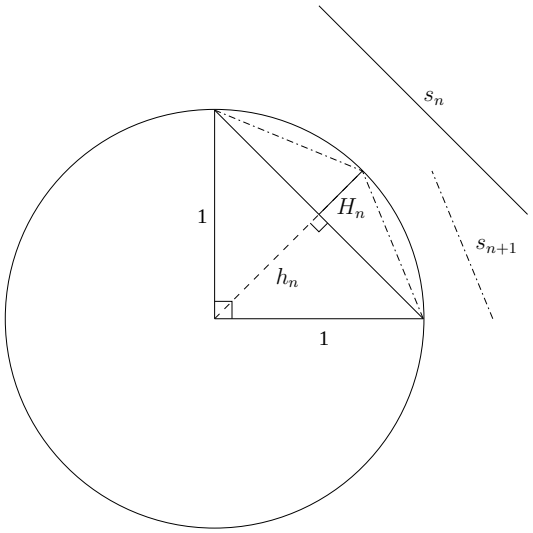


Abbildung 2: Definition der rekursiven Größen.

b) Es gilt mit den Definitionen aus Abbildung 2 (+Satz von Pythagoras!):

$$\begin{aligned} h_n^2 + \left(\frac{s_n}{2}\right)^2 &= 1 \\ H_n &= 1 - h_n \\ H_n^2 + \left(\frac{s_n}{2}\right)^2 &= s_{n+1}^2 \end{aligned}$$

und damit

$$\begin{aligned} s_{n+1} &= \sqrt{H_n^2 + \left(\frac{s_n}{2}\right)^2} = \sqrt{(1-h_n)^2 + \left(\frac{s_n}{2}\right)^2} \\ &= \sqrt{\left(1 - \sqrt{1 - \left(\frac{s_n}{2}\right)^2}\right)^2 + \left(\frac{s_n}{2}\right)^2} \\ &= \sqrt{\left(1 - 2\sqrt{1 - \left(\frac{s_n}{2}\right)^2} + 1 - \left(\frac{s_n}{2}\right)^2\right) + \left(\frac{s_n}{2}\right)^2} \\ &= \sqrt{2 - 2\sqrt{1 - \left(\frac{s_n}{2}\right)^2}} \;=\; \sqrt{2 - \sqrt{4 - s_n^2}}. \end{aligned}$$

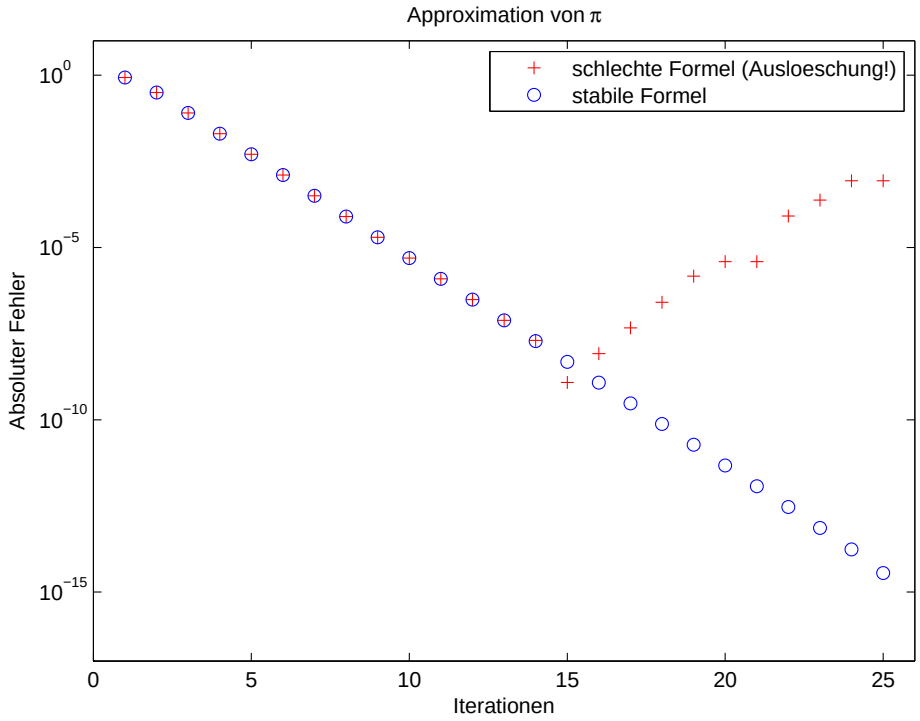
Der gesamte Umfang des 2^n-Ecks ergibt sich somit zu $U_n = 2^n \cdot s_n$ und damit also $\pi_n = U_n/2 = 2^{n-1} \cdot s_n$.

c) Effekt wie in Tabelle auf Angabe: Vermeintlich höhere Genauigkeit durch mehr Rekursionen verbessert das Ergebnis nicht sondern zerstört den gesamten Wert!
Problem: Auslöschung!

Algebraische Umformung zur Vermeidung der Auslöschung:

$$\begin{aligned} s_{n+1} &= \sqrt{2 - \sqrt{4 - s_n^2}} \;=\; \sqrt{2 - \sqrt{4 - s_n^2}} \cdot \frac{\sqrt{2 + \sqrt{4 - s_n^2}}}{\sqrt{2 + \sqrt{4 - s_n^2}}} \\ &= \frac{|s_n|}{\sqrt{2 + \sqrt{4 - s_n^2}}} \end{aligned}$$

Vergleich der absoluten Fehler der beiden Formeln in Bezug auf echte Lösung π in semilogarithmischer Skala (erstellt mit matlab-Programm archimedes.m aus www):



7) Ganzzahlen, Fest- und Gleitkommazahlen

Nr.	Frage		$A = I$	$A = F$	$A = G$
1	Darstellungsbeispiele	1 0000000	-0	-0,0	$-2^{+0} \cdot 2^{0}$
2		0 0000000	+0	+0,0	$2^{+0} \cdot 2^{0}$
3		0 1000000	64	8,0	$2^{-0} \cdot 2^{0}$
4		0 1000100	68	8,5	$2^{-1} \cdot 2^{0}$
5		0 1000110	70	8,75	$0,75 = 2^{-1} \cdot (2^{0} + 2^{-1})$
6		0 1000111	71	8,875	$0,875 = 2^{-1} \cdot (2^{0} + 2^{-1} + 2^{-2})$
7	Allg. Eigenschaften	$\lvert A \rvert$	256	256	256
8		$\max_{a \in A} a$	127	15,875	$57344 = 2^{15} \cdot (2^{0} + 2^{-1} + 2^{-2})$
9		$\min_{a \in A} a$	-127	-15,875	-57344
10		$\min_{a \in A, a>0} a$	1	0,125	$2^{-15} \cdot 2^{0}$
11		Ist 0 in A?	ja	ja	nein
12	Anzahl Zahlen von A im	$[2^{-15}, 2^{-14})$	0	0	4
13		$[1, 2)$	1	8	4
14		$[2, 4)$	2	16	4
15		$[4, 8)$	4	32	4
16		$[8, 16)$	8	64	4
17		$[16, 32)$	16	0	4
18		$[32, 64)$	32	0	4
19		$[64, 128)$	64	0	4
20		$[2^{15}, 2^{16})$	0	0	4
21		$[0, 1)$	1	8	$60 = (2^{4} - 1) \cdot 2^{2}$

Nr.	Frage		$A = I$	$A = F$	$A = G$
22	Rundungen: $\mathrm{rd}_A(x)$	3^{-5}	0	0	2^{-8}
23		2,1	2	2,125	2
24		3,1	3	3,125	3
25		9	9	9	8 (oder 10)
26		18	18	15,875	16 (oder 18)
27		1023	127	15,875	1024
28	Absoluter Rundungsfehler: $\lvert x - \mathrm{rd}_A(x)\rvert$	3^{-5}	3^{-5}	3^{-5}	$2{,}09{\cdot}10^{-4}$
29		2,1	0,1	0,025	0,1
30		3,1	0,1	0,025	0,1
31		9	0	0	1
32		18	0	2,125	2
33		1023	...	...	1
34	Relativer Rundungsfehler: $\left\lvert\frac{x-\mathrm{rd}_A(x)}{x}\right\rvert$	3^{-5}	1	1	0,051
35		$2,1$	0,048	0,012	0,048
36		$3,1$	0,032	0,008	0,032
37		9	0	0	0,111
38		18	0	0,118	0,111
39		1023	...	...	0,001

Beobachtungen

a) Betrachten Sie Ihre Antworten zu Fragen Nr. 2 und 3 für $A = G$. Was ist zu beobachten?

- Bei vorzeichenbehafteten Ganzzahlen ist 0 nicht eindeutig definiert.

b) Betrachten Sie Ihre Antworten zu Fragen Nr. 25 und 26 für $A = G$. Was ist zu beobachten?

- Ohne bestimmte Rundungsregeln gibt es Zahlen, die man nicht eindeutig runden kann.

c) Betrachten Sie Ihre Antwort zu Frage Nr. 25 für $A = G$. In manche Programmiersprachen (z.B. JavaScript) gibt es kein Format für Ganzzahlen. Welche Konsequenzen kann das haben?

- Es kann vorkommen, dass Rundungsfehlern auftreten, wenn man Objekte einfach zählen möchte.

d) Was ist die kleinste Ganzzahl, die bei 32-Bit IEEE Gleitkommazahlen nicht exakt darstellbar ist?

- $2^{24} + 1 \approx 17$ Millionen. Vergleichswerte: 1 Gigabyte $= 2^{30}$ Bytes,
  1 Mililiter Wasser enthält $\approx 2^{74}$ Moleküle.

e) Betrachten Sie die Antworten zu den Fragen 26.-27. für $A = F$. Welche Konsequenzen kann die Abwesenheit einer Inf-Darstellung haben?

- Man weiß nicht, ob 01111111_F genau 15,875 entspricht, oder etwas größerem.

f) Für eine Mantissendarstellung mit 2 Bits ist die Maschinengenauigkeit $\varepsilon_{Ma} = 2^{-3}$. Ver-

gleichen Sie ihre Antworten zu Fragen Nr. 34-39 mit der Maschinengenauigkeit. Ist

$$\left|\frac{x - \mathrm{rd}_G(x)}{x}\right| \leq \varepsilon_{Ma} \tag{1}$$

immer erfüllt?

- Die Rundungsfehler, die durch rd_G entstehen sind immer kleiner oder gleich ε_{Ma} (solange die Zahl im Range ist, natürlich). Man kann Gleichung (1) beweisen (betrachten Sie dazu Ihre Antworten zu Fragen Nr. 12-20.).

g) Gibt es eine Relation zwischen der Maschinengenauigkeit und der Zahl aus Frage Nr. 10?

- Nein. Die kleinste darstellbare positive Zahl ist nur von der Anzahl an Bits in den Exponenten abhängig. Die Maschinengenauigkeit ist nur von der Anzahl an Bits in der Mantisse abhängig.

h) Was ist der Anteil an Zahlen im $[0, 1)$ beim Format G (ungefähr)? Bei 32-Bit IEEE?

- Fast ein Viertel für beide. Bei G: $\frac{60}{256}$ =0,23.

i) Wie viele Zahlendarstellungen sind für spezielle Werte (Exponentenkombinationen $00\ldots0$ $11\ldots1$) bei IEEE reserviert?

- 2 (Vorzeichen) $\cdot$ 2 (Exponentenkombinationen) $\cdot$ 2^{23} (Mantissenmöglichkeiten) = $2^{25} = 33554432$ Zahlendarstellungen.

j) Gibt es einen Unterschied zwischen I und F? Betrachten Sie Ihre Antworten zu Fragen 12-21 für $A = I$ und $A = F$. Kann man F anhand I implementieren?

- Es gibt kaum Unterschiede. Man kann F durch I und einen Skalierungsfaktor implementieren.

k) Die darstellbare Zahlen bei I und F sind *linear* verteilt. Wie sind die Zahlen G verteilt?

- Logarithmisch.

l) Was hängt von der Anzahl an Bits in der Mantisse ab?

- Die Anzahl an darstellbare Zahlen im $[2^k, 2^{k+1})$ und die Maschinengenauigkeit ε_{Ma}.

m) Was hängt von der Anzahl an Bits in dem Exponent ab?

- Die Zahlen aus Fragen Nr. 8-10.