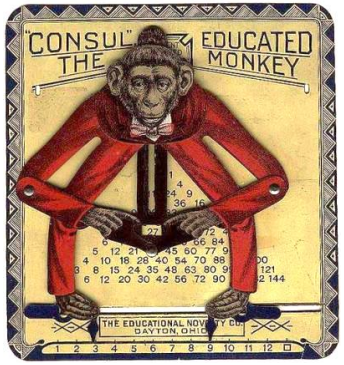
Einführung in die Programmierung



Tutorium 6

Wintersemester 2025/2026

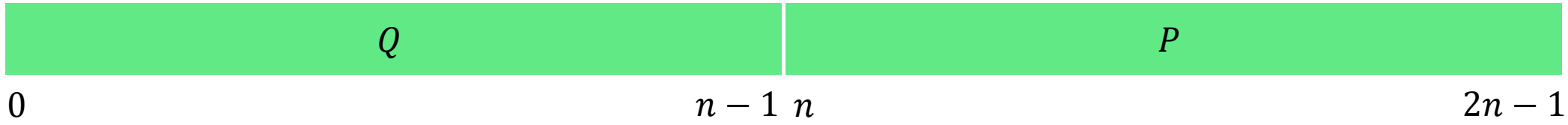
Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität
dortmund

Arbeitsgruppe
Systemsoftware

Naive Darstellung rationaler Zahlen im Rechner:

- Zähler $q = \sum Q_i \cdot 2^i$ und Nenner $p = \sum P_i \cdot 2^i$ werden hintereinander als gleichgroße Datenworte $D = Q_0 \cdots Q_{n-1} P_0 \cdots P_{n-1}$ abgelegt
- Vor dem Ablegen sind Nenner und Zähler mithilfe ihres größten gemeinsamen Teilers (ggT) zu kürzen
- **Problem**: Rechnen in diesem Zahlensystem ist aufwendig und kann schnell zu Überläufen im Zähler oder Nenner führen

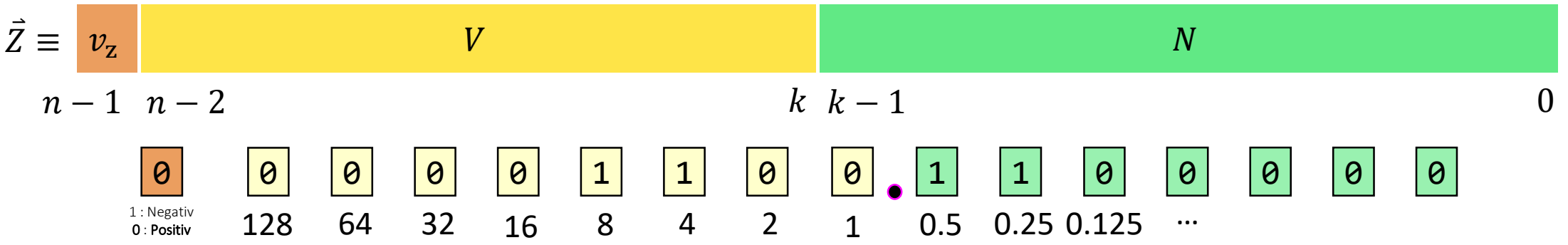


Darstellung von Zahlen im Festkommaformat (Naiv):

- Rationale Zahlen können wie gewohnt im Binärsystem mit zusätzlichem Nachkommateil dargestellt werden
- Wertigkeit des i-ten Bit im Vor- und Nachkommateil ergibt sich mit 2^{i-k}

$$z = (-1)^{v_z} \cdot (V_{n-2} \cdots V_k . N_{k-1} \cdots N_0)_2 \equiv (-1)^{v_z} \cdot 2^{-k} \cdot (V_{n-2} \cdots V_k N_{k-1} \cdots N_0)_2$$

$$= (-1)^{Z_{n-1}} \cdot 2^{-k} \cdot (Z_{n-2} Z_{n-3} \cdots Z_1 Z_0)_2 = (-1)^{Z_{n-1}} \frac{1}{2^k} \sum_{i=0}^{n-2} Z_i 2^i$$



Darstellung von Zahlen im Festkommaformat (Zweierkomplement):

- Festkommazahlen werden in realen Systemen üblicherweise im Zweierkomplement mit zusätzlichem Nachkommateil dargestellt
- Man rechnet mit ihnen genau wie mit ganzen Zweierkomplement-Zahlen

$$z = (V_{n-2} \cdots V_k . N_{k-1} \cdots N_0)_2 - v_z 2^{n-1} \equiv 2^{-k} \cdot [(V_{n-2} \cdots V_k N_{k-1} \cdots N_0)_2 - v_z 2^{n-1}]$$

$$= 2^{-k} \cdot [(Z_{n-2} Z_{n-3} \cdots Z_1 Z_0)_2 - Z_{n-1} 2^{n-1}] = \frac{1}{2^k} \sum_{i=0}^{n-1} (-1)^{\delta_{i,n-1} Z_i} 2^i$$

Kronecker-δ

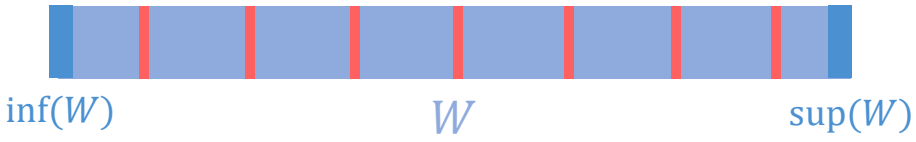
$$\delta_{i,j} \equiv \begin{cases} 1 \text{ if } i = j \\ 0 \quad \text{else.} \end{cases}$$

- **Fließkommazahlen** (**Gleitkommazahlen**, engl. **Floating-point numbers**) stellen rationale Zahlen durch die folgenden Bestandteile dar:
  - **Mantisse** M als normalisierte Binärzahl in gepackter, dh. impliziter Darstellung
  - **Kommaposition** als Charakteristik C in Exzess-Darstellung (**Exponent** E mit **Bias** k)
  - Vorzeichenbit v_z
- Bezeichnung für **Fließkommaformate**: SmEc

$$z = (-1)^{v_z} \cdot 2^{\overbrace{C-k}^{E}} \cdot (\underbrace{1}_{\text{Verstecktes Bit (engl. Hidden bit)}} . M)_2 = (-1)^{Z_{n-1}} \cdot 2^{\sum_{j=m}^{n-2}[Z_j 2^j] - (2^{\overbrace{n-m-1}^{c}-1} - 1)} \cdot \frac{1}{2^m} \underbrace{\sum_{i=0}^{m-1} Z_j 2^j}_{M}$$

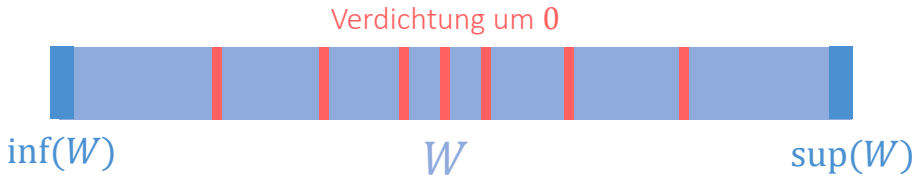
Zerlegungen und Verteilung der Fest- und Fließkommazahlen

(Festkommazahlen)



Äquidistante Zerlegung

(Fließkommazahlen)



Zerlegung in uneinheitliche Abstände

- Die **IEEE-754-Norm** umfasst drei unterschiedliche Gleitkommaformate
  - **Single-Precision-Format** (S23E8) mit Vorzeichenbit: $n = 1 + 8 + 23 = 32$
  - **Double-Precision-Format** (S52E11) mit Vorzeichenbit: $n = 1 + 11 + 52 = 64$
  - **Extended-Precision-Format** (S64E15) mit Vorzeichenbit: $n = 1 + 15 + 64 = 80$
- Darüber hinaus bietet IEEE-754 Möglichkeiten, auch ± 0 und $\pm \infty$ darzustellen: Darstellung der Null, wenn $E = 0$ und $M = 0$ sind
- Die Norm IEEE-754 wurde 1985 durch das **Institute of Electrical and Electronics Engineers** festgelegt

IEEE-754 Einfaches Präzisionsformat S23E8

Vorzeichen	Charakteristik	Mantisse	Bedeutung	Mathematische Darstellung
0	$000\cdots000$	M	Fließkommazahl ohne 1	$(0.M)_2 \cdot 2^{-127+1}$
1	$000\cdots000$	M	Fließkommazahl ohne 1	$-(0.M)_2 \cdot 2^{-127+1}$
0	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$(1.M)_2 \cdot 2^{C-127}$
1	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$-(1.M)_2 \cdot 2^{C-127}$
0	$111\cdots111$	$000\cdots000$	Unendlich (positiv)	inf bzw. ∞
1	$111\cdots111$	$000\cdots000$	Unendlich (negativ)	-inf bzw. $-\infty$
0	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1}_{+} \cdot (1.M)$
1	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1}_{-} \cdot (1.M)$

IEEE-754 Doppeltes Präzisionsformat S52E11

Vorzeichen	Charakteristik	Mantisse	Bedeutung	Mathematische Darstellung
0	$000\cdots000$	M	Fließkommazahl ohne 1	$(0.M)_2 \cdot 2^{-1023+1}$
1	$000\cdots000$	M	Fließkommazahl ohne 1	$-(0.M)_2 \cdot 2^{-1023+1}$
0	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$(1.M)_2 \cdot 2^{C-1023}$
1	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$-(1.M)_2 \cdot 2^{C-1023}$
0	$111\cdots111$	$000\cdots000$	Unendlich (positiv)	inf bzw. ∞
1	$111\cdots111$	$000\cdots000$	Unendlich (negativ)	-inf bzw. $-\infty$
0	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1}_{+} \cdot (1.M)$
1	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1}_{-} \cdot (1.M)$

IEEE-754 Erweitertes Präzisionsformat S64E15

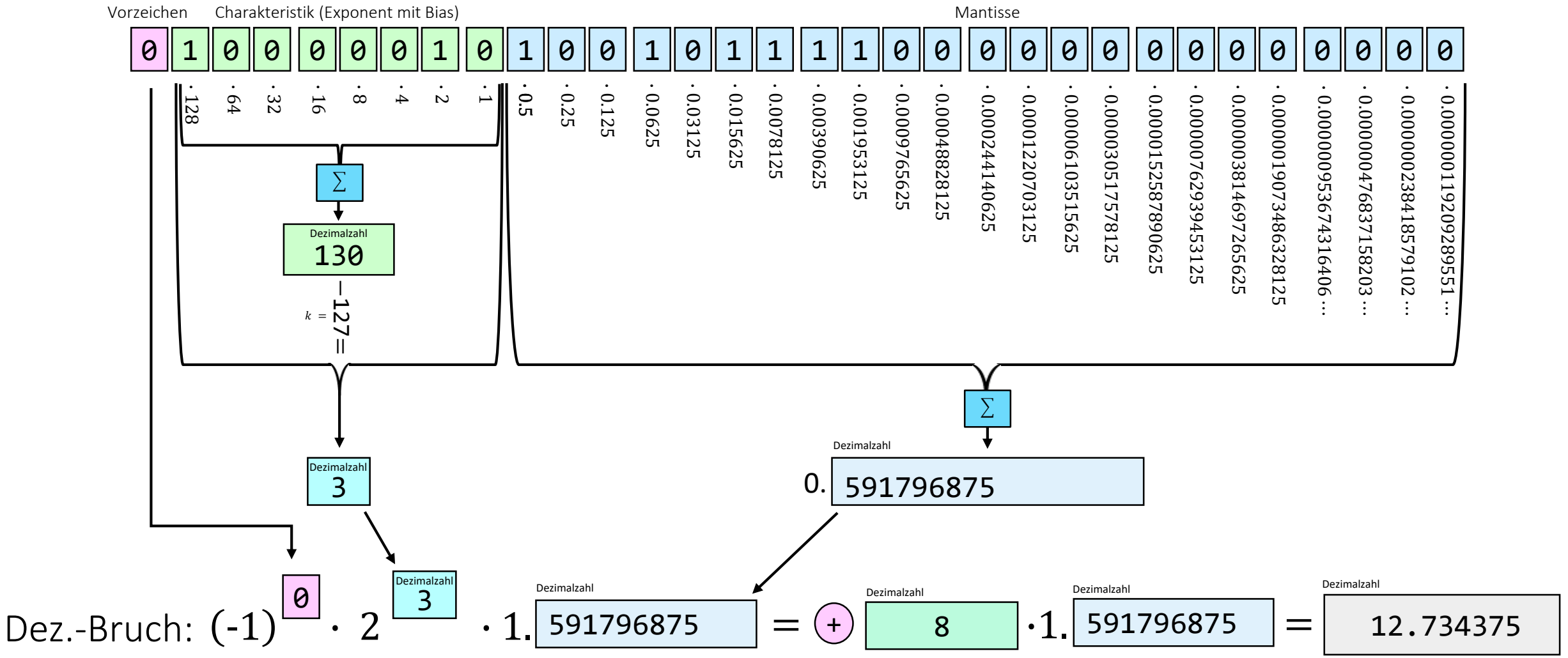
Vorzeichen	Charakteristik	Mantisse	Bedeutung	Mathematische Darstellung
0	$000\cdots000$	M	Fließkommazahl ohne 1	$(0.M)_2 \cdot 2^{-16383+1}$
1	$000\cdots000$	M	Fließkommazahl ohne 1	$-(0.M)_2 \cdot 2^{-16383+1}$
0	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$(1.M)_2 \cdot 2^{C-16383}$
1	$0 < E < 2^{c-1} - 1$	M	Fließkommazahl	$-(1.M)_2 \cdot 2^{C-16383}$
0	$111\cdots111$	$000\cdots000$	Unendlich (positiv)	inf bzw. ∞
1	$111\cdots111$	$000\cdots000$	Unendlich (negativ)	-inf bzw. $-\infty$
0	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1} \cdot (1.M)$
1	$111\cdots111$	$M \neq 0$	Keine (rationale) Zahl	N. a. N. z.B.: $\sqrt[2]{-1} \cdot (1.M)$

Datentypen für Fließkommazahlen in **C** und **C++**

Art der Daten	Schlüsselwort	Format	Speicherplatz	Platzhalter
Rationale Zahlen $\mathbb{Q}$	`float`	Single-Precision-Format **S8E32**	32 Bit ≙ 8 Nibble ≙ 4 Byte	%f
	`double`	Double-Precision-Format **S11E52**	64 Bit ≙ 16 Nibble ≙ 8 Byte	%lf
	`long double`	Extended-Precision-Format **S15E64**	80 Bit ≙ 20 Nibble ≙ 10 Byte	%Lf

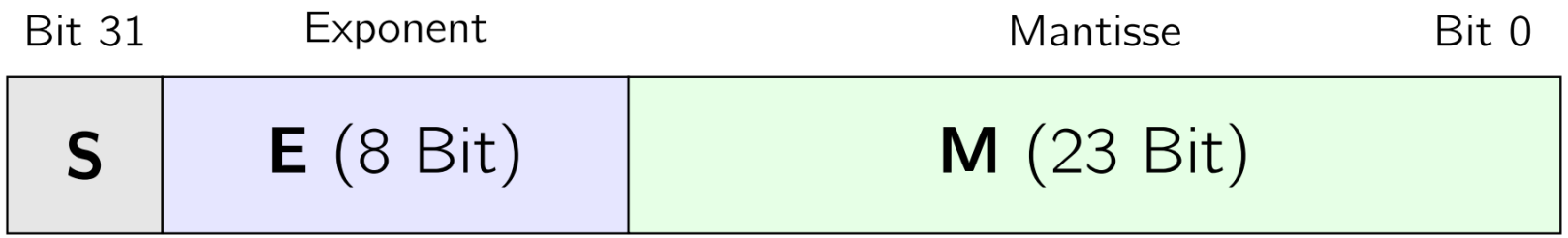
- **Fließkomma-Verarbeitungseinheit** (engl. **Floating-Point-Unit**, kurz: **FPU**) ist auf **Fließkommazahlenarithmetik** spezialisierte Hardware
- Bereits 1980 nutzte die FPU 8087 von Intel® das **S15E64**-Format

Lesen einer **S23E8**-Fließkommazahl (Lösung)



Implementieren Sie die Funktion extract_parts, um eine gegebene 32-Bit-Zahl u in ihre Bestandteile zu zerlegen. Nutzen Sie dazu die definierten Masken (SIGN_MASK, EXP_MASK, MANT_MASK) und Shift-Operatoren:

- Isolieren Sie das Vorzeichenbit in die Variable sign.
- Extrahieren Sie die 8 Exp-Bits und schieben Sie diese an die niederwertigste Stelle (exp).
- Extrahieren Sie die 23 Mantissen-Bits (mant).



Code

```
40  static void print_float_parts(uint32_t u, const char *label, float as_decimal) {
41      uint32_t sign = 0, exp = 0, mant = 0;
42
43      // a)
44      sign = (u & SIGN_MASK) >> 31;
45      exp  = (u & EXP_MASK) >> 23;
46      mant = (u & MANT_MASK);
47
48      printf("%s = %g\n\r", label, (double)as_decimal);
49      printf("Sign bit of %s: %u\n\r", label, sign);
50      printf("Exponent of %s: ", label);
51      print_bits_uint(exp, EXP_BITS);
52      putchar('\n');
53      putchar('\r');
54      printf("Mantissa of %s: ", label);
55      print_bits_uint(mant, MANT_BITS);
56      putchar('\n');
57      putchar('\n');
58      putchar('\r');
59  }
```

Berechnen Sie das Vorzeichen per XOR sowie den Exponenten unter Berücksichtigung des Bias und multiplizieren Sie die Mantissen inklusive Hidden Bit. Normalisieren Sie das entstehende Produkt und führen Sie abschließend die Rundung „Round to Nearest, Ties to Even“ durch, wobei auch ein möglicher Überlauf nach dem Runden abgefangen werden muss.

$$x \cdot y = (-1)^{v_{Z_x}} \cdot 2^{\overbrace{C_x - k}^{E_x}} \cdot (1.M_x)_2 \cdot (-1)^{v_{Z_y}} \cdot 2^{\overbrace{C_y - k}^{E_y}} \cdot (1.M_y)_2$$

$$= (-1)^{v_{Z_x} \oplus v_{Z_y}} \cdot 2^{\overbrace{C_x - k}^{E_x} + \overbrace{C_y - k}^{E_y}} \cdot \mathrm{norm}(1.M_x \cdot 1.M_y)_2$$

$$= (-1)^{\overbrace{v_{Z_x} \oplus v_{Z_y}}^{v_{Z_{xy}}}} \cdot 2^{\overbrace{C_x + C_y - k - k}^{E_{xy}}} \cdot 2^{\mu} \cdot \frac{1}{2^{2m}} \overbrace{M_x \cdot M_y}^{M_{xy}}$$

Multiplikation zweier Fließkommazahlen – Code (1)

Code

```
61  static uint32_t ieee32_mul_u(uint32_t a, uint32_t b) {
62      int32_t res_exp = 0;
63      uint32_t res_sign = 0;
64      uint64_t prod = 0;
65
66      if (is_nan_u(a) || is_nan_u(b)) return 0x7FC00000;
67      if (is_zero_u(a) || is_zero_u(b)) {
68          if (is_inf_u(a) || is_inf_u(b)) return 0x7FC00000; // 0 * Inf
69          return (a ^ b) & SIGN_MASK;
70      }
71
72      if (is_inf_u(a) || is_inf_u(b)) return ((a ^ b) & SIGN_MASK) | EXP_MASK;
73
74      // a) Vorzeichen bestimmen
75      res_sign = (a ^ b) & SIGN_MASK;
76
77      // b) Exponenten addieren
78      res_exp = (int32_t)((a & EXP_MASK) >> 23) + (int32_t)((b & EXP_MASK) >> 23) - EXP_BIAS;
79
80      ...
```

Multiplikation zweier Fließkommazahlen – Code (2)

Code

```
80      // c) Mantissen inkl. Hidden Bit holen
81      uint64_t mant_a = (a & MANT_MASK) | HIDDEN_BIT;
82      uint64_t mant_b = (b & MANT_MASK) | HIDDEN_BIT;
83
84      // d) 48 Bit Produkt berechnen
85      prod = mant_a * mant_b;
86
87      // e) Entscheiden, ob man um 23 oder 24 Bit shiftet
88      int shift;
89      if (prod & (1ULL << 47)) { // Ergebnis >= 2.0
90          shift = 24;
91          res_exp++;
92      } else {
93          shift = 23;
94      }
95      ...
```

Multiplikation zweier Fließkommazahlen – Code (3)

Code

```
96      // f) Extrahieren der Bits, die weggeworfen würden (Rest)
97      uint64_t remainder = prod & ((1ULL << shift) - 1);
98      uint64_t half      = 1ULL << (shift - 1); // Das Bit genau bei 0.5
99
100     prod >>= shift; // Normalisieren
101
102     // Round to nearest, ties to even
103      if (remainder > half) {
104         prod++; // Aufrunden
105     } else if (remainder == half) {
106         if (prod & 1) prod++; // Bei genau 0.5 zur geraden Zahl runden
107     }
108     ...
```

Multiplikation zweier Fließkommazahlen – Code (3)

Code

```
109     // g) Falls durch das Aufrunden ein Überlauf in der Mantisse passiert (z.B. alle 1er
        werden zu 100...)
110     if (prod & (1ULL << 24)) { // Sollte nach Shift max 23 Bits haben (Bit 24 wäre Hidden
        für nächste Stufe)
111         prod >>= 1;
112         res_exp++;
113     }
114
115     if (res_exp >= 255) return res_sign | EXP_MASK;
116     if (res_exp <= 0) return res_sign; // Underflow -> 0
117
118     return res_sign | (res_exp << 23) | (prod & MANT_MASK);
119     }
```

Implementieren Sie die Funktion ieee32_inv_u, um das Reziproke 1 / x zu berechnen, indem Sie den neuen Exponenten über 253 – exp bestimmen und die neue Mantisse durch ganzzahlige Division von 2.0 durch die Eingangsmantisse ermitteln. Runden Sie das Ergebnis anschließend korrekt ("Round to Nearest, Ties to Even") und normalisieren Sie es bei Bedarf.

$$z^{-1} = \frac{1}{(-1)^{v_z} \cdot 2^{C-k} \cdot (1\,.\,M)_2} = (-1)^{v_z} \cdot 2^{\overbrace{(2k-C)-k}^{-E}} \cdot 2^{m} \cdot 1.0/M$$

Reziproke einer Fließkommazahl – Code (1)

Code

```
222  // a) Zerlegung der Eingabe
223  sign = x & SIGN_MASK;
224  exp  = (int32_t)((x & EXP_MASK) >> 23);
225  mant = (x & MANT_MASK) | HIDDEN_BIT;
226
227  // Sonderfälle (IEEE 754)
228  if (is_nan_u(x)) return 0x7FC00000;          // NaN -> Quiet NaN
229  if (is_zero_u(x)) return sign | EXP_MASK;    // +/- 0 -> +/- Inf
230  if (is_inf_u(x)) return sign;                // +/- Inf -> +/- 0
231     ...
```

Reziproke einer Fließkommazahl – Code (2)

Code

```
232    // b) Berechnung des neuen Exponenten
233    // Formel: 1 / (M * 2^E) = (1/M) * 2^-E.
234    // Wir berechnen (2/M) * 2^(-E-1) um im Bereich [1.0, 2.0) zu bleiben.
235    // Neuer Bias-Exp = 127 + (- (exp - 127)) - 1 = 253 - exp.
236    res_sign = sign;
237    res_exp  = 253 - exp;
238    ...
```

Reziproke einer Fließkommazahl – Code (3)

Code

```
239    // c) Division der Mantisse (2.0 / M)
240    // Wir nutzen einen 64-Bit Zähler (1ULL << 47), der 2.0 repräsentiert.
241    // Division: (2^47) / mant. Ergebnis liegt um 2^24 (Bit 24 gesetzt wenn >= 2.0,
       sonst Bit 23).
242    numerator = 1ULL << 47;
243    res_mant = numerator / mant;
244    remainder = numerator % mant;
245    ...
```

Reziproke einer Fließkommazahl – Code (4)

Code

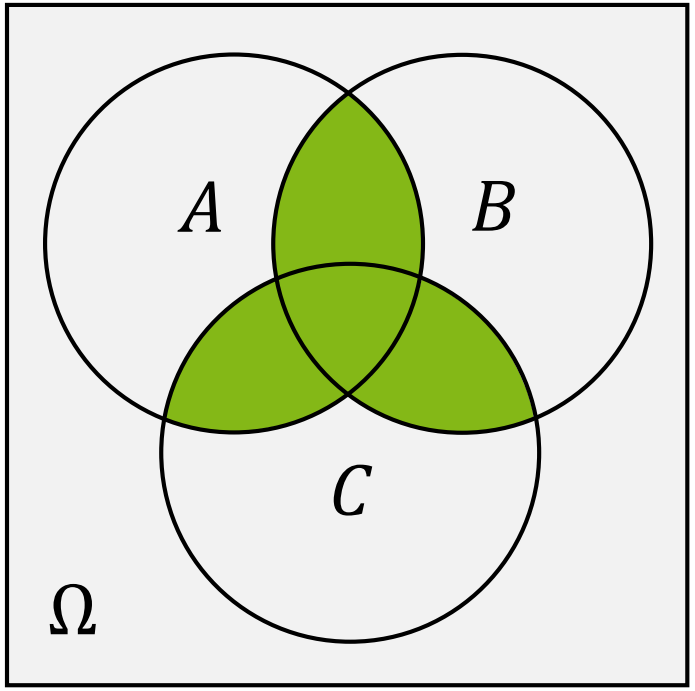
```
246    // d) Rundung (Round to Nearest, Ties to Even)
247    // Ist der Rest größer als der halbe Divisor? -> Aufrunden
248    // Ist der Rest genau halb und das aktuelle Ergebnis ungerade? -> Aufrunden
       (zur geraden Zahl)
249    if (remainder > (mant / 2)) {
250        res_mant++;
251    } else if ((remainder * 2 == mant) && (res_mant & 1)) {
252        res_mant++;
253    }
254    ...
```

Reziproke einer Fließkommazahl – Code (4)

Code

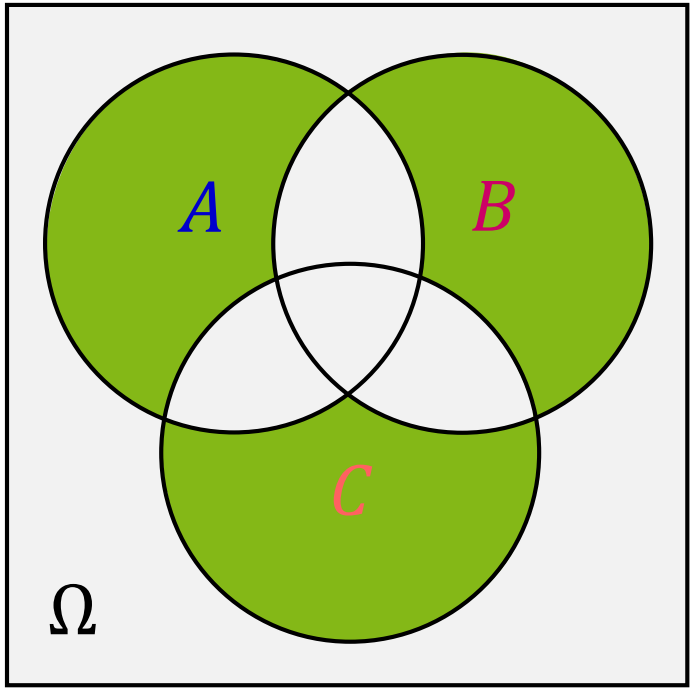
```
246    // e) Normalisierung nach Division
247    // Falls x exakt 1.0 (oder 2^k) war, ist 2.0/1.0 = 2.0.
248    // Dann ist Bit 24 gesetzt (1ULL << 24). Wir müssen durch 2 teilen und Exp erhöhen.
249    if (res_mant & (1ULL << 24)) {
250        res_mant >>= 1;
251        res_exp++;
252    }
253    ...
```

Venn-Diagramm $a)$



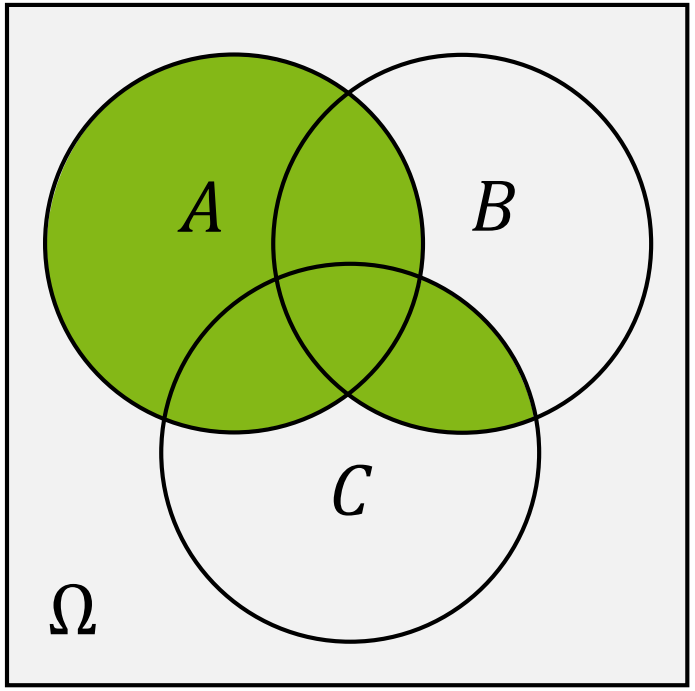

```
(A&B)|(A&C)|(B&C)
```

Venn-Diagramm $b)$

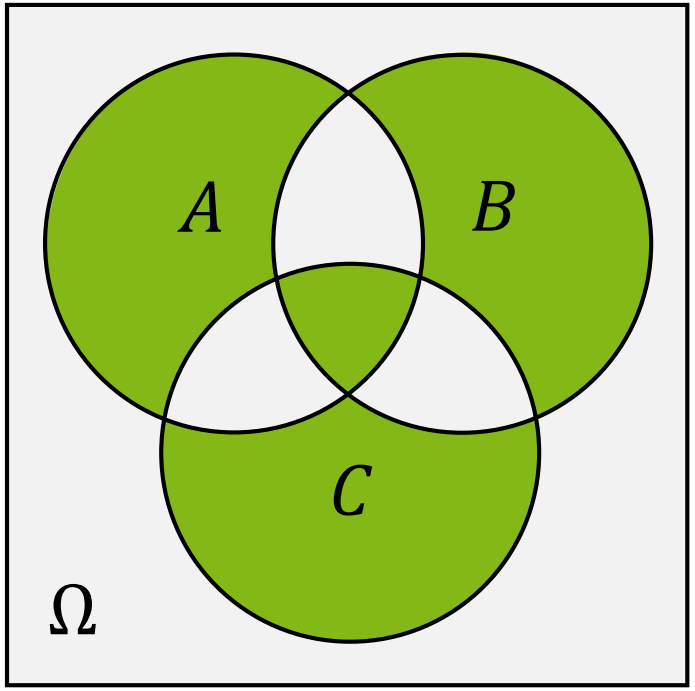



```
(A&~B&~C)|(~A&B&~C)|(~A&~B&C)
```

Venn-Diagramm $c)$




```
A | (B & C)
```



A ^ B ^ C