OTH Regensburg

Studiengang
IN, IT B. Sc.
Prof. Dr. Martin Weiß
Wintersemester 2024/25

Computerarithmetik und Rechenverfahren

1. Übungsblatt: Zahlen und ihre Darstellung

1.1. Wiederholen Sie aus der Vorlesung Mathematik 1: Vektoren, Matrizen, insb. Multiplikation von Vektoren und Matrizen, sowie Lösbarkeit von linearen Gleichungssystemen mit quadratischer Systemmatrix.

1.2. Umwandlung und Rechnen mit b-adischen Zahlen Wandeln Sie die Dezimal-Zahlen 765 und 254 in das Binär-, Oktal- und Hexadezimalsystem und addieren Sie die Zahldarstellungen bzgl dieser Basis. Wandeln Sie die Ergebnisse ins Dezimalsystem zurück.

1.3. Umwandlung in Binärzahlen mit Vorzeichen Geben Sie die bit-Darstellung folgender ganzer Zahlen nach einer Zuweisung

`x =` v `;`

an für die jeweiligen C/C++-Datentypen (auf Intel-Architektur, also little endian, `char` mit 8 bit, `short` mit 16 bit, `int` bzw. `long` mit 32 bit; wie bei Microsoft Visual C/C++ oder GNU C auf Intel-Architektur):

a) $v = 10$ an `unsigned char x`, `unsigned short x`, `short x`,

b) $v = -10$ an `char x`, `short x`

c) $v = -1$ an `char x`, `short x`

d) $v = 255$ an `unsigned char x`, `unsigned short x`, `short x`.

1.4. Rechnen mit b-adischen Zahlen Lösen Sie folgende Rechenaufgaben *ohne* Umwandlung ins Dezimalsystem oder ein anderes Zahlensystem:

a) $\texttt{00101001}_2 + \texttt{01101011}_2 = ?$; Ergebnis in 8-bit Zweierkomplement

b) $\texttt{00101001}_2 - \texttt{01101011}_2 = ?$; Ergebnis in 8-bit Zweierkomplement

c) $\texttt{00AF}_{16} + \texttt{00FE}_{16} = ?$; Summanden und Ergebnis als 16-bit `short`

1.5. Ganze Zahlen in MATLAB Erklären Sie die Werte `int16(40000)`, `uint16(40000)`, `int16(80000)`. Was passiert bei

```
a = int16(1024);
a = a * int16(1024);
disp(a);
disp(int16(a));
```

im Vergleich zu

```
a = 1024;
a = a * 1024;
disp(a);
disp(int16(a));
```

und im Vergleich zum Hochsprachen-Code

```
short int a = 1024;
a *= 1024;
printf("%d\n", a);
```

1.6. Gleitpunktaddition

a) Schreiben Sie die Dezimalzahlen 173.251 und 3210.798 als normalisierte Gleitpunktzahlen mit 5 Nachkommastellen und perfekter Rundung (round to even) im Dezimalsystem.

b) Berechnen Sie mit Gleitpunktarithmetik (5 Stellen für die Mantisse) die Summe der beiden Zahlen aus a).

c) Schreiben Sie die Zahlen $\frac{5}{6}$, $\frac{3}{4}$ als normalisierte Gleitpunktzahlen zur Basis 2 mit 5 Stellen Mantisse und perfekter Rundung (round to even) und berechnen Sie die Gleitpunktsumme in dieser Darstellung. Vergleichen Sie mit dem exakten Wert im Dezimalsystem.

1.7. Assoziativgesetz gilt nicht bei Gleitkommazahlen Es sei die Gleitpunkt-Addition mit Mantisse der Länge 3 im Dezimalsystem

$$x \oplus y = fl(x+y).$$

Man zeige, daß für $a = 0.333, b = 0.117 * 10^1, c = -0.1 * 10^1$:

$$(a \oplus b) \oplus c \neq a \oplus (b \oplus c),$$

d.h. diese arithmetische Operation ist nicht assoziativ.

1.8. Alte Klausuraufgabe Die Fließkommazahlen nach IEEE 754-Standard mit einfacher Genauigkeit $x_1, \ldots, x_4$ seien in `float`-Variablen `x1, x2, x3, x4` gespeichert. Ihr Wert als Bitmuster sei gegeben durch:

Variable	Byte 1	Byte 2	Byte 3	Byte 4
x1	00101011	00000000	01111100	00000000
x2	00110101	00000000	01111100	00000000
x3	11111000	10110000	00000000	00000000
x4	01000000	10110000	00000000	00000000

a) Welchen Wert als bit-Muster haben die Variablen `y1, y2, y3`, nachdem folgende Anweisungen durchlaufen wurden? Stellen, die mit $*$ markiert sind, müssen Sie nicht angeben.

```
float y1, y2, y3;
y1 = -x1 / 8;
y2 = 22.5;
y3 = x3*x3;
```

Variable	Byte 1	Byte 2	Byte 3	Byte 4
y1				********
y2				********
y3				********

b) Welchen Wert hat x4 als reelle Zahl? ____

1.9. Gegeben seien Fließkomma-Zahlen in einfacher und doppelter Genauigkeit in einem C/C++-Programm:

```
float fx, fy, fz;
double dx, dy, dz;
```

Die bit-Repräsentation nach IEEE 745 sei gegeben durch (Trennstriche jeweils nach 8 bit, das niedrigswertige Bit steht rechts):

Variable								
fx	00111111	01000000	00000000	00000000				
fy	00000000	00000000	00000000	00000000				
fz	00111111	01000000	00000000	00000000				
dx	00000011	10010000	00000000	00000000	00110000	00000000	00000000	00000000
dy	01000111	01000000	00000000	00000000	11001100	00000000	00000000	00000000
dz	01111111	11101110	00000000	00000000	00000000	00000000	00000000	00000000

Geben Sie, ohne explizite Umwandlung in reelle Zahlen, die Werte der mit ☐ markierten Bits der sechs Variablen an, nachdem folgendes Programm bearbeitet wurden:

```
fx = dx;
fy = dy;
fz = 2*fz;
dx = 1.0;
dy = dy / (-8.0);
dz = 1024 * dz;
```

Variable								
fx		******	********	******				
fy		******	********	******				
fz		******	********	******				
dx			********	********	********	********	********	********
dy			********	********	********	********	********	********
dz		****	********	********	********	********	********	********

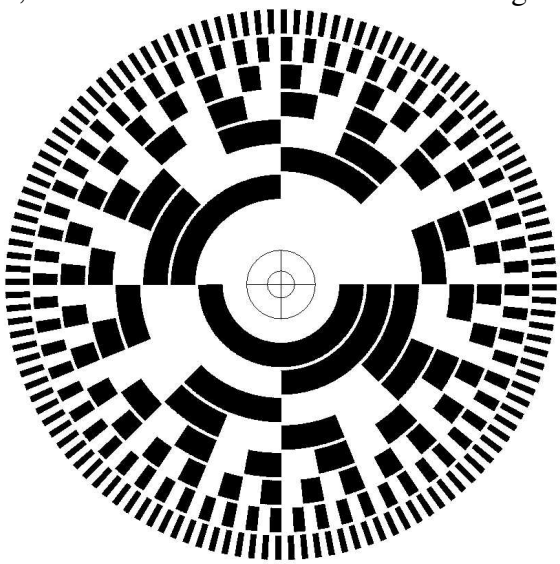
1.10. ** Division mit Rest In C/C++ liefern die Operatoren / und % nur für $x \geq 0$, $b > 0$ die Vorzeichenbedingung $r \geq 0$!

$$x = mb + r, \qquad m \in \mathbb{N}_0, r \in \{0, 1, \ldots, b-1\}$$

Was ist das Ergebnis folgender Operationen? Leiten Sie eine Regel ab!

```
int x = 0, b = 0;
x = 10; b = 7;
printf("x = %3d b = %3d  x mod b = %3d\n", x, b, x % b);
printf("x = %3d b = %3d  x div b = %3d\n", x, b, x / b);
x = 10; b = -7;
printf("x = %3d b = %3d  x mod b = %3d\n", x, b, x % b);
printf("x = %3d b = %3d  x div b = %3d\n", x, b, x / b);
x = -10; b = 7;
printf("x = %3d b = %3d  x mod b = %3d\n", x, b, x % b);
printf("x = %3d b = %3d  x div b = %3d\n", x, b, x / b);
x = -10; b = -7;
printf("x = %3d b = %3d  x mod b = %3d\n", x, b, x % b);
printf("x = %3d b = %3d  x div b = %3d\n", x, b, x / b);
```

1.11. Positionserfassung mit Gebersystem Bei einem Motor wird die Position hardware-seitig von einem Gebersystem (Encoder) mit einer 12-bit-unsigned-Zahl erfasst, also sog. single-turn-Zähler: innerhalb einer Umdrehung wird eine absolute Position z.B. nach dem Prinzip einer bit-Codierung auf einer Drehscheibe erfasst, die multi-turns werden von einer übergeordneten Steuerung gezählt.



Die Steuerung beschreibt die antriebsseitige Position intern mit einer vorzeichenbehafteten 64-bit-Zahl. Der Motor wird mit einer Untersetzung von 1:200 auf den Abtrieb übersetzt, und die Abtriebsposition in der HMI als Winkel in Grad in einer Fließkommazahl mit einfacher Genauigkeit dargestellt.

Wieviele Umdrehungen kann der Motor in eine Richtung machen, bevor ein Überlauf bzw. eine Bereichsüberschreitung

a) in der Anzeige erfolgt?

b) in der internen Darstellung erfolgt?

Wie lange dauert das jeweils, wenn ein Elektromotor ständig mit seiner Maximaldrehzahl von etwa 3000 U/min betrieben würde?

1.12. ** Welche Ausgabe erwarten Sie bei diesen C-Anweisungen?

```
void infnan()
{
   float x1 = 1e38, x2 = 2, x3 = -10, x4 = 0, x5 = 0;
   float y[10];

   y[1] = x1 * x1;
   y[2] = y[1] * x2;
   y[3] = x3 * x1;
   y[4] = x4 / x5;
   y[5] = log(x4);
   y[6] = sqrt(x3);

   for (int i = 1; i <= 6; ++i)
   {
      // nicht jedes System kennt isinf(), isnan()
      // printf("y[%d] = %f     Inf %d    NaN  %d\n",
                    d, y[d], isinf(y[i]), isnan(y[i]));
      printf("y[%d] = %f \n", i, y[i]);
   }
   printf("%d\n", y[1] == y[2]);
   printf("%d\n", y[4] == y[5]);
   printf("%d\n", y[4] == y[6]);
}
```

1.13. ** Darstellung von Sonderwerten

Schreiben Sie eine C++-Klasse, mit der Sie IEEE-Zahlen und deren bit-Darstellung ausgeben können. Prüfen Sie, welche Codierung Sie erhalten für Operationen, die den Körper $\mathbb{R}$ verlassen, etwa:

$$\frac{1.0}{0.0}, \qquad \frac{-1.0}{0.0}, \qquad \frac{0.0}{0.0}, \qquad \log(0.0), \qquad \log(-1.0), \qquad \sqrt{-1.0}$$

1.14. ** Darstellung der 0

Im IEEE-Standard gibt es eine positive und eine negative 0 mit Darstellungen:

```
0 00000000 00000000000000000000000
1 00000000 00000000000000000000000
```

Beim Zweierkomplement für ganze Zahlen hatten wir 2 Darstellungen für den Wert 0 als nachteilig angesehen. Es gibt auch andere Meinungen, siehe Wikipedia:

The IEEE 754 standard for floating-point arithmetic (presently used by most computers and programming languages that support floating point numbers) requires both $+0$ and -0. Real arithmetic with signed zeros can be considered a variant of the extended real number line such that $1/-0 = -\infty$ and $1/+0 = +\infty$; division is only undefined for $\pm 0/\pm 0$ and $\pm\infty/\pm\infty$.

Negatively signed zero echoes the mathematical analysis concept of approaching 0 from below as a one-sided limit, which may be denoted by $x \to 0-$, $x \to 0-$, or $x \nearrow 0$. The notation -0 may be used informally to denote a small negative number that has been rounded to zero. The concept of negative zero also has some theoretical applications in statistical mechanics and other disciplines.

It is claimed that the inclusion of signed zero in IEEE 754 makes it much easier to achieve numerical accuracy in some critical problems,[1] in particular when computing with complex elementary functions.[2] On the other hand, the concept of signed zero runs contrary to the general assumption made in most mathematical fields that negative zero is the same thing as zero. Representations that allow negative zero can be a source of errors in programs, if software developers do not take into account that while the two zero representations behave as equal under numeric comparisons, they yield different results in some operations.

Testen Sie in Ihrer Programmiersprache, wie Sie die 2 Werte anlegen können, und ob die Werte bei Vergleichen als gleich angesehen werden oder nicht!

1.15. Sonderwerte `Inf`, `NaN`

Welche Ausgabe erwarten Sie bei diesen MATLAB-Anweisungen? Was erwarten Sie bei entsprechenden Anweisungen in C-basierten Sprachen?

```
result = [0/0 1/0 0/1 -1/0 0/-1]
Inf == -Inf
Inf == Inf
NaN == NaN
Inf == 2*Inf
```

1.16. Sonderwerte positive und negative Null

In MATLAB kann man über das Vorzeichen eine positive oder negative Darstellung der 0 erhalten. Praktisch wenig Nährwert, da min nicht auf negative 0 testen kann: es gilt [0.0 == -0.0] (anders bei `NaN`: `NaN==NaN` liefert immer `false`).

Sie können aber prüfen, dass die bit-Darstellung unterschiedlich ist, indem Sie positive und negative 0 zuweisen, und die Variablen aus dem workspace in eine Datei schreiben.

```
x = 0.0
save zeropositive.mat x

x = -0.0
save zeronegative.mat x
```

In python / numpy gibt es auch entsprechend definierte Konstanten, siehe etwa `https://numpy.org/doc/stable/reference/constants.html`: `numpy.NZERO`, `numpy.PZERO`.

1.17. Wieviele Zahlen gibt es?

a) Wieviele verschiedene Zahlen aus $\mathbb{N}_0$ gibt es als `uint32`-Werte?

Wieviele verschiedene Zahlen aus $\mathbb{Z}$ gibt es als `int32`-Werte?

b) Wieviele verschiedene Zahlen aus $\mathbb{Q}$ gibt es als `float`-Werte? Wieviele verschiedene Zahlen aus $\mathbb{R}$ gibt es als `float`-Werte?

D.h. Sonderwerte `NaN`, `Inf` werden nicht gezählt!

Dasselbe für `double`?

c) Die Funktion `rand` liefert eine Zufallszahl aus dem Intervall $(0,1)$. Wieviele verschiedene Zahlen (= mögliche Zufallszahlen) gibt es als `double`-Werte in diesem Intervall?

Beachten Sie: Zufallszahlen auf dem Rechner sind i.d.R. *Pseudozufallszahlen*, d.h. sie werden ausgehend von einem Startwert (*seed*) x_0, der von aussen gesetzt werden kann, über eine deterministische, aber schwer vorhersehbare Funktion als Folge $x_{k+1} = f(x_k)$ aus der vorhergehenden Pseudozufallszahl erzeugt.

d) [***] Bestimmen Sie die Periode der von `rand` erzeugten Folge x_k, ausgehend vom – immer gleichen – Startwert nach Hochlauf von MATLAB. Mit `rng` können Sie den Zufallszahlengenerator zurücksetzen.

e) [*** Sinnloses Wissen für die nächste Party] Um Größen zu veranschaulichen, finden Sie in vielen populärwissenschaftlichen Texten Vergleiche mit der Anzahl der Sterne im Universeum, der Atome im Universum, der Zeit in Sekunden seit dem Urknall, der Staatsverschuldung Griechenlands bzw. der USA. Ermitteln Sie diese Werte im Internet. Welche Datentypen reichen, um die Werte *exakt* zu speichern?

1.18. ** Denormalisierte Zahlen

Der IEEE-Standard sieht inzwischen auch *denormalisierte Zahlen* vor. Bei Charakteristik 0 können die Mantissenbits verwendet werden, um betraglich kleinere Zahlen als diejnigen der Form $x = sb^e(1+\sum_{j=1}^{m} d_j b^{-j})$ darzustellen, nämlich Zahlen der Form $x = sb^e(\mathbf{0}+\sum_{j=1}^{m} d_j b^{-j})$.

```
void denormalisiert()
{
    char BinaryRepresentation[NUM_DOUBLE_BITS+1];
    // kleinste normalisierte float-Zahl
    float x = convertBinToFloat("00000000100000000000000000000000");

    convertFloatToBin(x, BinaryRepresentation);
    printf("    binary:  x = %s   %e %lf\n\n", BinaryRepresentation, x, log(x)/log(2.0))

    for (int i = 0; i < 24; ++i)
    {
        x /= 2;
        convertFloatToBin(x, BinaryRepresentation);
        printf("%2d: binary:  x = %s   %e %lf\n", i,
            BinaryRepresentation, x, log(x)/log(2.0));
    }
}
```

Erklärung: Die kleinste normalisierte `float`-Zahl hat den Wert 2^{-126}, vgl. Tabellen über Kenngrößen im IEEE-Format, und die bit-Darstellung in der Aufteilung Vorzeichen, Charakteristik, Mantisse:

```
0 00000001 00000000000000000000000
```

Dividiert man diese Zahl durch 2, erhält man die denormalisierte Zahl 2^{-127} mit bit-Darstellung

```
0 00000000 10000000000000000000000
```

Also: $2^{-126} \cdot 0.10000000000000000000000_2$.

Die kleinste positive denormalisierte Zahl ist 2^{-149} mit bit-Darstellung

```
0 00000000 00000000000000000000001
```

MATLAB-Code dazu:

```
xMin = realmin('single')
eMin = round(log(xMin)/log(2))
xDenormMin = x/(2^23)
eDenormMin = round(log(xDenormMin)/log(2))
xDenormMin/2
```

1.19. IEEE

Bestimmen Sie die IEEE-754-Darstellung von `y1`, `y2`, `y3`, nachdem folgende Anweisungen durchlaufen wurden:

```
float x1 = 3.25;
float x2 = 1.5;
float y1 = x1 * 16;
float y2 = x1 + x2;
float y3 = 1.0 / 7.0;
```

Die Darstellung von `x1`, `x2` ist zur Vereinfachung gegeben:

	Byte 1	Byte 2	Byte 3	Byte 4
x1	01000000	01010000	00000000	00000000
x2	00111111	11000000	00000000	00000000

1.20. Halbe Genauigkeit ist nicht genug (1021.4+262.4)*.27

1.21. Halbe Genauigkeit ist nicht genug

Quelle: `https://www.golem.de/news/machine-learning-arm-intel-und-nvidia-standardisieren-8html`

In einer branchenübergreifenden Kooperation haben die Hardware-Hersteller ARM, Intel und Nvidia ein neues Format für Gleitkommazahlen vorgestellt, die nur noch eine Genauigkeit von 8 Bit (FP8) aufweisen sollen. Bisher offiziell von der IEEE standardisiert sind Gleitkommazahlen mit 16 Bit (halbe Genauigkeit), 32 Bit (einfache Genauigkeit), 64 Bit (doppelte Genauigkeit) und größere. Sie wissen: 80 bit extended (1 bit Vorzeichen, 15 bit Charakteristik / Exponent, 63 bit fraction), quadruple, octuple format.

Der als wissenschaftliche Ausarbeitung formulierte Vorschlag des Standards umfasst zwei grundlegende Kodierungen der neuen 8-Bit-Gleitkommazahlen: E4M3 (4 Bit Exponent and 3 Bit Mantisse) sowie E5M2 (5 Bit Exponent and 2 Bit Mantisse). Hinzu kommt wie bei den IEEE-Standards ein Vorzeichen-Bit sowie festgelegte Kodierungen für Unendlich, NaN, Null, sowie den Bereich von normalisierten und subnormalen Zahlen.

Beantworten Sie für die zwei vorgeschlagenen Formate:

a) Wieviele normalisierte Zahlen gibt es? Zählen sie +0 und -0 als eine Zahl.

b) Wieviele denormalisierte Zahlen gibt es?

c) Wie lautet die betragsgrößte bzw. betragskleinste Zahl (normalisiert) binär und dezimal?

d) Wie lauten kleinster und größter Exponent bzw. Charakteristik? Was ist der bias? Geben Sie eine Formel für den bias als Funktion der Bit-Zahl der Charakteristik an.

e) Was ist die Maschinengenauigkeit?

1.22. Ermitteln Sie aktuelle Werte im Internet, und ordnen Sie die Zahl-Anteile in den angegebenen Einheiten aufsteigend nach an:

- Staatsverschuldung Deutschlands [€]
- Staatsverschuldung Griechenlands [€]
- Staatsverschuldung der USA [$]
- Entfernung Sonne – Erde [m]

- Masse der Erde [kg]