Grundlagen der Informatik

Prof. Dr. J. Schmidt
Fakultät für Informatik

GDI – WS 2018/19

Zahlendarstellung – Binäres Rechnen 2

Hochschule Rosenheim
University of Applied Sciences

Leitfragen 2.4

- Welche charakteristischen Merkmale weist die binäre Multiplikation und Division auf?
- Wie werden in digitalen Rechenanlagen „reelle" Zahlen dargestellt?

Binäre Multiplikation (1)

- Rechenregeln für die Multiplikation zweier binärer Zahlen
  - $0 \cdot 0 = 0$
  - $0 \cdot 1 = 0$
  - $1 \cdot 0 = 0$
  - $1 \cdot 1 = 1$

Identisch mit den Regeln der logischen UND-Verknüpfung

- Multiplikation mehrstelliger Zahlen
  - Multiplikation des Multiplikanden mit den einzelnen Stellen des Multiplikators
  - Stellenrichtige Addition der Zwischenergebnisse

- Beispiel: Aufgabe 10 · 13

```
1 0 1 0 · 1 1 0 1
-------------------
      1 0 1 0
        1 0 1 0
          0 0 0 0
            1 0 1 0
-------------------
    1 0 0 0 0 0 1 0
```

Ergebnis: 130

Binäre Multiplikation (3)

- Beispiel: Aufgabe 17.375 · 9.75

```
1 0 0 0 1.0 1 1 · 1 0 0 1.1 1
-------------------------------
      1 0 0 0 1 0 1 1
            1 0 0 0 1 0 1 1
              1 0 0 0 1 0 1 1
                1 0 0 0 1 0 1 1
-------------------------------
      1 0 1 0 1 0 0 1.0 1 1 0 1
```

Zwischenergebnis 1 0 1 0 1 0 0 1.0 1 1 0 1

Stellenrichtiges Einfügen des Kommas

Ergebnis: 169.40625$_{(10)}$

Binäre Division (1)

- Rechenregeln für die Division zweier binärer Zahlen
  - 0 : 0 = nicht definiert
  - 0 : 1 = 0
  - 1 : 0 = nicht definiert
  - 1 : 1 = 1
- Durchführung der binären Division analog dem im Zehnersystem üblichen Verfahren

- Beispiel 20 : 6

```
1 0 1 0 0  :  1 1 0  =  1 1.0 1 0 1 ...
 -1 1 0
 1 0 0 0
  -1 1 0
     1 0 0 0
      -1 1 0
         1 0 0 0
          -1 1 0
               ...
```

Ergebnis: 3.333...$_{(10)}$

Realisierung im Rechner (1)

- Multiplikation wird mittels wiederholter Addition durchgeführt
- Division wird mittels wiederholter Subtraktion durchgeführt
- **Sonderfall**:
  Multiplikator oder Divisor sind eine Zweierpotenz 2^k
  - Multiplikation bzw. Division kann einfacher und schneller erfolgen
  - Durch Verschiebung von entsprechend vielen Bits (k) nach links bzw. rechts
  - Bei $2^{\mathbf{1}}$ um **1** Bit, $2^{\mathbf{2}}$ um **2** Bits, bei $2^{\mathbf{3}}$ um **3** Bits usw.

Realisierung im Rechner (2)

- Beispiele
  - $13 \cdot 4$

    `1101 · 100 = 110100`

				1	1	0	1
			1	1	0	1	0
		1	1	0	1	0	0

2 Bits nach links

= 52

  - $20 \cdot 8$

    `10100 · 1000 = 10100000`

    3 Bits nach links

  - $20 : 4$

    `10100 : 100 = 101`

    2 Bits nach rechts

  - $26 : 4$

    `11010 : 100 = 110` (Rest 2) (→ evtl. Informationsverlust!)

    2 Bits nach rechts

Realisierung im Rechner (3)

- Ablage der zu verschiebenden Zahl in einem dem Rechenwerk direkt zugeordneten Speicherplatz (Register, Akkumulator)
- Register besitzen meist Übertrags-Bit (Carry)

- Beispiel:
  Verschieben $1101_{(2)}$ um 1 Stelle nach rechts

Register	Carry
0 0 0 0 1 1 0 1	0
0 0 0 0 0 1 1 0	1

Übertrags-Bit wird von 0 auf 1 gesetzt (Setzen eines Flag)

„Reelle“ Zahlen

- In Programmiersprachen wird anstatt des im Deutschen üblichen Kommas der **Punkt** verwendet, um ganzzahligen Teil vom gebrochenen Teil einer „reellen“ Zahl abzutrennen

- Zwei unterschiedliche Typen
  - Festkommazahlen (fixed point)
  - Gleitkommazahlen (floating point)

Festkommazahlen (1)

- Punkt (das Komma) steht immer an einer bestimmten festgelegten Stelle
  - wobei der Punkt nicht eigens mitgespeichert wird

$$Z = z_{n-1} z_{n-2} \cdots z_1 z_0 \,.\, z_{-1} z_{-2} \cdots z_{-m\,(2)}$$

$$Z = \sum_{i=-m}^{n-1} z_i \cdot 2^i$$

  - Z hat die Länge $n + m$
    n Stellen vor dem und **m** Stellen nach dem Komma

Festkommazahlen (2)

- Nachteile der Festkomma-Darstellung
  - Mit einer bestimmten Anzahl von Bits kann nur ein beschränkter Wertebereich abgedeckt werden
  - Die Stelle des Punkts (Kommas) muss allgemein festgelegt werden

    (Wo soll man diesen festlegen, wenn manchmal mit sehr kleinen, hochgenauen Werten und ein anderes Mal mit sehr großen Werten gearbeitet werden muss?)
- Festkommadarstellung wird nur in Rechnern verwendet, die für Spezialanwendungen benötigt werden
  - sonst Gleitkommadarstellung

Gleitkommazahlen (1)

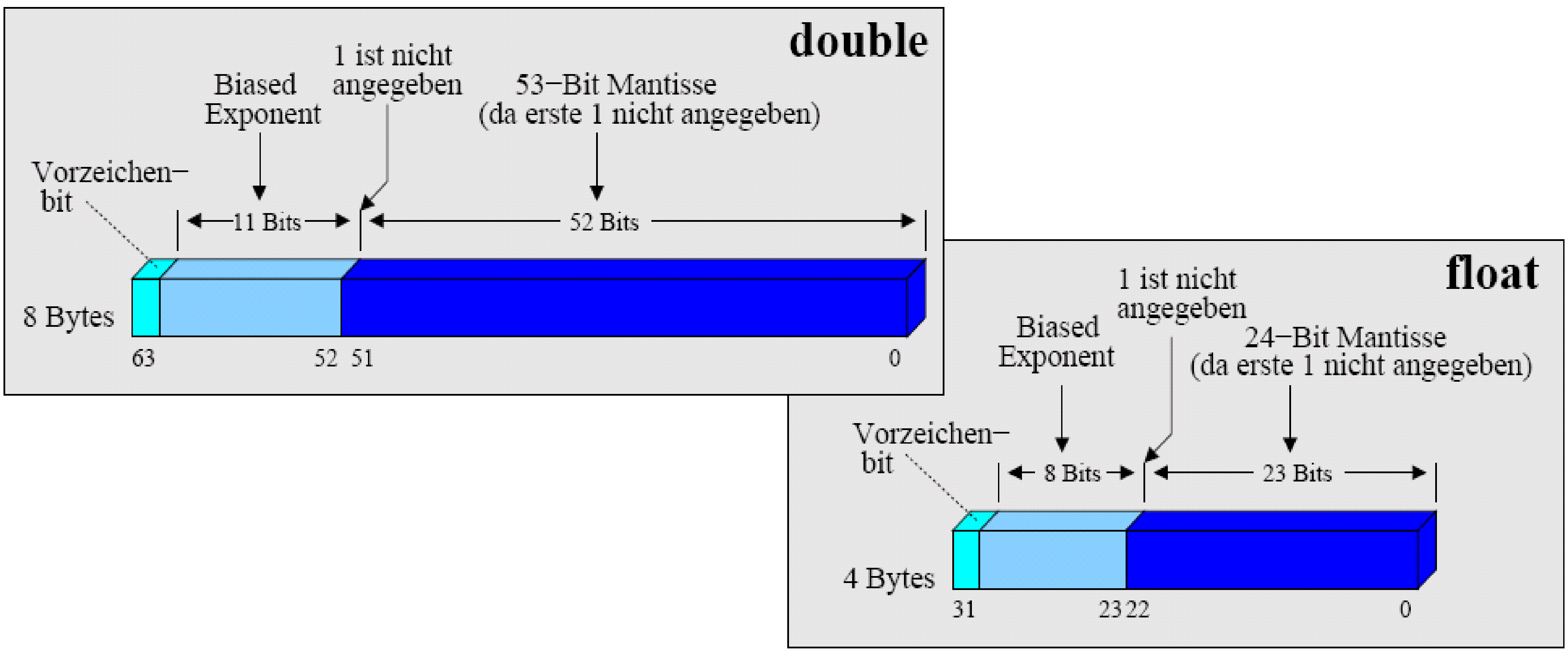
- Jede reelle Zahl kann beispielsweise in folgender Form angegeben werden

$$2.3756 \cdot 10^3$$

- Zwei Bestandteile
  - **Mantisse** (2.3756) und
  - **Exponent** (3), der ganzzahlig ist
- Wird in den meisten Rechnern verwendet
  - Aber nicht Basis 10, sondern Basis 2
- Die für die Darstellung verwendete Anzahl von Bytes legt fest, ob man mit
  - einfacher (Datentyp **`float`**) oder mit
  - doppelter Genauigkeit (Datentyp **`double`**) arbeitet

Gleitkommazahlen (2)

- IEEE-Format (binär, Norm IEEE 754-2008)
  - C/C++- und Java-Datentypen verwenden vier Bytes für `float` und acht Bytes für `double`
  - In der Norm werden auch Darstellungen für halbe (16 Bit) und vierfache Genauigkeit (128 Bit) definiert



Gleitkommazahlen (2)

- IEEE-Format
  - geht von normalisierten Gleitkommazahlen aus

- Normalisierung
  - der Exponent wird so verändert, dass
  - der gedachte Dezimalpunkt immer rechts von der ersten Nicht-Null-Ziffer liegt
    (im Binärsystem ist dies immer eine 1)

Gleitkommazahlen (3)

- Beispiel
  - $17.625_{(10)}$

    $= 16 + 1 + \frac{1}{2} + \frac{1}{8}$
    $= 10001.101_{(2)}$
    $= 10001.101 \cdot 2^0$

  - Normalisierte Form
    - Schiebe Dezimalpunkt hinter die erste signifikante Ziffer
    - Passe Exponent entsprechend an

$$\boxed{= 1.0001101 \cdot 2^4}$$

Gleitkommazahlen (4)

- Aus Normalisierung (IEEE-Format) ergibt sich
  - In der Mantisse steht das höchstwertige „Einser-Bit“ immer links vom gedachten Dezimalpunkt
    - Außer für 0.0 und einige andere Sonderfälle
    - Dieses Bit wird nicht gespeichert
  - Exponent ist eine ganze Zahl, welche (nach Addition eines Bias) ohne Vorzeichen dargestellt wird
    - Durch Bias-Addition wird für den Exponent keine Vorzeichenrechnung benötigt (immer positiv)
    - Wert von Bias hängt vom Genauigkeitsgrad ab
      - float (mit 4 Bytes, 8 Bits für Exponent): Bias = 127
      - double (mit 8 Bytes, 11 Bits für Exponent): Bias = 1023
  - Vorzeichenbit zeigt das Vorzeichen der Mantisse
    - Mantisse immer als Betragswert
    - auch im negativen Fall nicht als Komplement

- Beispiel
  - 17.625 ($1.0001101 \cdot 2^4$)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - Biased Exponent: 10000011 = 131
  - Bias: 01111111 = 127
  - Wirklicher Exponent: 00000100 = 4

Gleitkommazahlen (6)

- Nach IEEE gilt für `float` (einfach) und `double` (doppelt)

	Einfach	Doppelt
Vorzeichen Bits	1	1
Exponenten Bits	8	11
Mantissen Bits	23	52
Bit insgesamt	32	64
BIAS	127	1023
Exponentenbereich	[-126, 127]	[-1022, 1023]

Gleitkommazahlen (7)

- Null
  - positive (+0.0) und negative (–0.0) Null
    - bei Vergleichsoperationen sind beide gleich
  - Verwendung
    - Darstellung der Null
    - Rundung auf +/– 0.0 bei Unterlauf (Loch um die Null)
  - Darstellung
    - (biased) Exponent = 0
    - Mantisse = 0

Gleitkommazahlen (8)

- Unendlich (INF)
  - plus (+∞) und minus (–∞) Unendlich
  - Verwendung
    - Zahlen, deren Betrag zu groß ist um noch dargestellt werden zu können (Überlauf)
    - Rechnungen, die per Definition Unendlich ergeben (z.B. Division einer Zahl z≠0 durch Null: z / 0.0 = ∞)
  - Darstellung
    - (biased) Exponent = 111….1
    - Mantisse = 0

- keine Zahl (NaN)
  - Verwendung
    - Darstellung ungültiger Werte
    - Rechnungen, die undefinierte Ergebnisse liefern
      - 0.0 / 0.0 = NaN
      - ∞ / ∞ = NaN
      - $\sqrt{-3}$ = NaN
  - Darstellung
    - (biased) Exponent = 111….1
    - Mantisse > 0

- Geben Sie für die Zahl 125.875 die `float`-Darstellung (nach IEEE-Format) an.

1. Bestimmung Bias = $127_{(10)}$
2. Umwandlung in duale Festkommazahl ohne Vorzeichen
   - Anteil vor Komma: $125_{(10)}$ $= \mathbf{1111101}_{(2)}$
   - Anteil nach Komma: $0.875_{(10)}$ $= \mathbf{0.111}_{(2)}$

     $0.875 \cdot 2 = 1.75 \rightarrow \mathbf{1}$

     $0.75 \cdot 2 = 1.5 \rightarrow \mathbf{1}$

     $0.5 \cdot 2 = 1.0 \rightarrow \mathbf{1}$
3. Normieren

   $\mathbf{1111101.111} \cdot 2^0 = \mathbf{1.111101111} \cdot 2^6$
4. Berechnung dualer Exponent

   $2^6 \rightarrow 6_{(10)} + \text{Bias} = 133_{(10)}$

   $= \mathbf{10000101}_{(2)}$
5. Bestimmung Vorzeichen Bit

   Positiv $\rightarrow$ **0**
6. Zusammensetzen

V	Exponent	Mantisse
0	**10000101**	**11110111100000000000000**

Ungenauigkeit von Gleitkommazahlen (1)

- Gleitkommazahlen, die im Dezimalsystem genau darstellbar sind, können im Dualsystem nicht immer genau dargestellt werden
  - Daraus resultieren Ungenauigkeiten
  - Merke: `float`- und `double`-Werte sollte man niemals auf Gleichheit prüfen!
- Beispiel
  - Ausgeben der Zahlen 0.1 bis 1.0 mit einer Schrittweite von 0.1
  - `float`- oder `double`-Variable als Laufvariable bei Iterationen

Ungenauigkeit von Gleitkommazahlen (2)

- Endlosschleife, da die Abbruchbedingung nie erreicht wird

```c
#include <stdio.h>

int  main(void)
{
   float  i = 0.1;

   while (i != 1.0)
   {
      printf("%.10f\n", i);
      i = i + 0.1;
   }
   return 0;
}
```

Ungenauigkeit von Gleitkommazahlen (3)

- Gewünschtes Ergebnis wird erreicht

```c
#include <stdio.h>

const float EPSILON = 1e-6;

int  main(void)
{
    float  i = 0.1;

    while (i <= 1.0+EPSILON)
     {
       printf("%.10f\n", i);
       i = i + 0.1;
     }
    return 0;
}
```

Ungenauigkeit von Gleitkommazahlen (4)

- Bessere Methode:
  Umgehen der Ungenauigkeit durch Arbeiten mit ganzzahligen Laufvariablen

```
#include <stdio.h>

int  main(void) {
   int  i = 1;

   while (i <= 10)
    {
      printf("%.10f\n", (float)i/10);
      i = i + 1;
    }
   return 0;
}
```

Ungenauigkeit von Gleitkommazahlen (5)

- Gleitkommaarithmetik ist nicht assoziativ!
  - $(u + v) + w \neq u + (v + w)$
    $(u * v) * w \neq u * (v * w)$
- …und auch nicht distributiv
  - $u * (v + w) \neq (u * v) + (u * w)$
- Beispiel (dezimal, 8 Stellen genau)
  - (11111113. + (-11111111.)) + 7.5111111 =
    2.0000000 + 7.5111111 = 9.5111111
  - 11111113. + (-11111111. + 7.5111111) =
    11111113. + (-11111103.) = 10.000000