Teil 2

Floating point - arith - decimal add

- 1. Anpassung der Dezimalpunkte (Zahl mit kleineren exponent „shiften“ bis exponent gleich)
- 2. Addition der significands
- 3. Normalisieren
- 4. Runden

- Beispiel (4-Stellen):
  9.999 * 10^1 + 1.610 *10^-1
  - 1.:
    9.999 * 10^1 + 0.01610 *10^1
  - 2.:
    9.999+ 0.01610=10.015
    ->10.015 *10^1
  - 3.:
    1.0015 *10^2
  - 4.:
    1.002 *10^2

Floating point - arith - decimal add - Übung

- 1. Anpassung der Dezimalpunkte (Zahl mit kleineren exponent „shiften“ bis exponent gleich)
- 2. Addition der significands
- 3. Normalisieren
- 4. Runden

- Beispiel (4-Stellen):
  2.320 *10^1 + 7.325 *10^0
  - 1.:
  - 2.:
  - 3.:
  - 4.:

Floating point - arith - binary add

- 1. Anpassung der Dezimalpunkte (Zahl mit kleineren exponent „shiften“ bis exponent gleich)
- 2. Addition der significands
- 3. Normalisieren
- 4. Runden

- Beispiel (4-Stellen):
  dezimal: 0.5 + −0.4375
  binär: 1.000 *2^-1 + −1.110 *2^-2
  - 1.:
    1.000 *2^-1 + -0.111 *2^-1
  - 2.:
    1.000 + -0.111=0.001
    ->0.001 *2^-1
  - 3.:
    1.000 *2^-4
  - 4.:
    1.000 *2^-4 (hier keine Änderung durch Runden)

    //dec: 1*1/16 -> 0.0625

Floating point - arith - binary add - Übung

- 1. Anpassung der Dezimalpunkte (Zahl mit kleineren exponent „shiften“ bis exponent gleich)
- 2. Addition der significands
- 3. Normalisieren
- 4. Runden

- Beispiel (5-Stellen):
  dezimal: 23.25 + 7.375
  binär: 101.1101*2^2 + 111.011 *2^0
  - 1.:
  - 2.:
  - 3.:
  - 4.:

Floating point - arith - decimal mult

- 1. Addition der Exponenten
- 2. Multiplikation der significands
- 3. Normalisieren
- 4. Runden
- 5. Bestimmung des Vorzeichens

- Beispiel (4-Stellen):
  1.110 *10^10 * 9.200 *10^-5
  - 1.:
    10+ -5=5
  - 2.: 1.110 * 9.200

```
1110 * 9200
    9990
     2220
         0
          0
    11
--------------
   10212000
->10.212
->10.212*10^5
```

  - 3.:
    1.0212*10^6
  - 4.:
    1.021*10^6
  - 5:
    +1.021*10^6

Floating point - arith - decimal mult - Übung

- 1. Addition der Exponenten
- 2. Multiplikation der significands
- 3. Normalisieren
- 4. Runden
- 5. Bestimmung des Vorzeichens

- Beispiel (4-Stellen): 2.320*10^1 * -7.325*10^0
  - 1.:
  - 2.:
  - 3.:
  - 4.:
  - 5.:

Floating point - arith - binary mult

- 1. Addition der Exponenten
- 2. Multiplikation der significands
- 3. Normalisieren
- 4. Runden
- 5. Bestimmung des Vorzeichens

- Beispiel (4-Stellen):
  dezimal: 0.5 * -0.4375
  binär: 1.000 *2^-1 * -1.110 *2^-2
  - 1.:
    -1+ -2=-3
  - 2.:
```
1.000 * -1.110
1110 * 1000
    1110
        0
         0
          0

--------------
    1110000
-> 1.110
-> 1.110*2^-3
```
  - 3.:
    1.110*2^-3 (hier keine Änderung)
  - 4.:
    1.110*2^-3 (hier keine Änderung)
  - 5:
    -1.110*2^-3 //"+"*"-" -> "-"
    //dec: ->-0.21875

Floating point - arith - binary mult - Übung

- 1. Addition der Exponenten
- 2. Multiplikation der significands
- 3. Normalisieren
- 4. Runden
- 5. Bestimmung des Vorzeichens

- Beispiel (8-Stellen, truncate):
  dezimal: 3.25 * –6.5
  binär: 11.01 *2^0 * -11.01 *2^1
  - 1.:
  - 2.:
  - 3.:
  - 4.:
  - 5:

Floating point - rounding (Q)

- nicht alle gebrochenen Zahlen innerhalb der Wertegrenzen sind mit floating point darstellbar
- Beispiel
  - darstellbar: 0.5
  - darstellbar: 12345
  - nicht darstellbar: 0.1

- Rounding = Bestimme eine geeignete darstellbare Zahl x^* für eine möglicherweise nicht darstellbare Zahl x.
  - x^* kann eine der beiden darstellbaren Zahlen x^+, x^- sein, für die gilt:

$$x^- \leq x \leq x^+$$

  - ***verschiedene*** Rundungsregeln sind möglich

Floating point - rounding - decimal

- Beispiel: Runden auf einen glatten Euro Betrag

Rule	1,50 €	2,50 €	- 1,50 €
Round-to-even	2 €	2 €	- 2 €
Round-toward-zero	1 €	2 €	- 1 €
Round-down	1 €	2 €	- 2 €
Round-up	2 €	3 €	- 1 €

Data rep - floating point - rounding - „Round-to-even“ (B)

- Warum ist round-to-even sinnvoll?
  - systematisches Runden (entweder nach oben oder nach unten) kann zu einem statistischen Fehler (*bias*, Verschiebung) führen

- Veranschaulichung im Dezimalbereich:
  - Folgende Messwerte werden gerundet. Danach wird der Mittelwert bestimmt

    {2.5; 3.5; 4.5; 5.5}

  - wirklicher Mittelwert: 16 / 4 = 4.0 = 4
  - Mittelwert bei round-down 14 / 4 = 3.5 = 3
  - Mittelwert bei round-up: 18 / 4 = 4.5 = 5
  - Mittelwert bei round-to-even: **16 / 4 = 4.0 = 4**

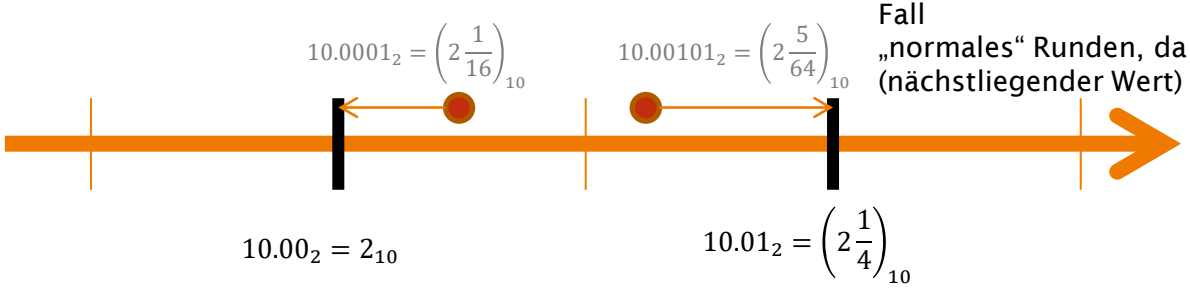
Floating point - rounding - binär 1/2

- $b_i, i = 0 \ldots n$ sind darstellbare Binärziffern
- b_0 ist die nach dem Runden niederwertigste (least significant) Bitziffer, -> Runden bedeutet: Bestimmung von b_0
- für $b_n \ldots b_{m+1}.b_m \cdots b_0 0 \cdots$ klar -> abrunden
  und $b_n \ldots b_{m+1}.b_m \cdots b_0 1 \cdots 1$ klar -> aufrunden
- unklar nur $b_n \ldots b_{m+1}.b_m \cdots b_0 1000 \cdots 0$

Rule	$\cdots b_1[b_0 = 0]100 \cdots > 0$	$\cdots b_1[b_0 = 0]100 \cdots < 0$
Round-to-even	$\cdots b_1\,0000 \cdots$	$\cdots b_1\,0000 \cdots$
Round-toward-zero	$\cdots b_1\,0000 \cdots$	$\cdots b_1\,0000 \cdots$
Round-down	$\cdots b_1\,0000 \cdots$	$\cdots b_1\,1000 \cdots$
Round-up	$\cdots b_1\,1000 \cdots$	$\cdots b_1\,0000 \cdots$

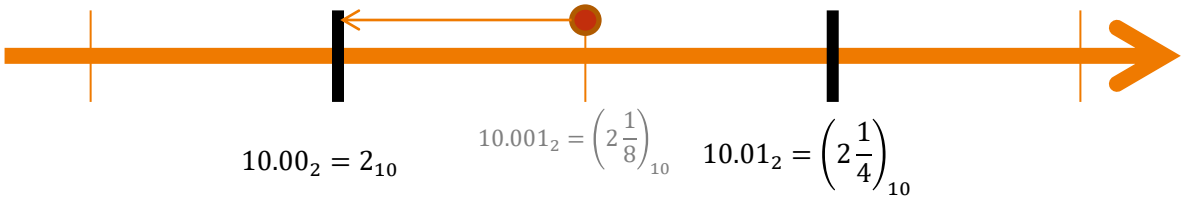
Floating point - rounding - binär 2/2 - Beispiel

- Beispiel runden auf zwei Stellen nach Binärpunkt



Fall
„normales“ Runden, da klar
(nächstliegender Wert)

Unklarer Fall genau in der Mitte:
hier Runden mit
„Round-to-even“
letzte Stelle gerade, also 0
($b_0 \rightarrow 0$)



Floating point - arith - precision 1/3 (R)

- Was passiert eigentlich wirklich beim Rechnen mit floating point Werten?

- Für eine beliebige Berechnung wird definiert: $x \circledcirc y$
  - D.h. man ersetzt $\circledcirc$ z.B. durch $+, -, \times$ oder $/$

- Das Ergebnis der Berechung ist $\mathrm{round}\,(x \circledcirc y)$
  - D.h. für $x \circledcirc y$ wird das **mathematisch exakte** Ergebnis der Operation eingesetzt

Data rep - floating point - arith 2/3 - Beispiel (R)

- Beispiel 1:

```
(3.14 + 1e10) – 1e10 =
round ( round ( 3.14 + 1e10) – 1e10) =
round (1e10 – 1e10) = 0
```

- Beispiel 2:

```
3.14 + (1e10 – 1e10) =
round ( 3.14 + round ( 1e10 – 1e10)) =
round ( 3.14 + 0) = 3.14
```

Data rep - floating point - arith 3/3 (R)

- Folgerung:
  - Es ist also nicht vernachlässigbar, in welcher Reihenfolge floating point Operationen durchgeführt werden
  - -> d.h. das Assoziativgesetz ist bei floating point nicht mehr gültig!

- Beispiel:

```
x = a + b + c;
y = b + c + d;
```

```
t = b + c;
x = a + t;
y = t + d;
```

- Anwendung/Auswirkungen:
  - Bestimmte Optimierungen dürfen für floating point nicht durchgeführt werden! (Wissenschaftliche Software/ Compilerbau)