Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

HTA-FR – Kurs Telekommunikation

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

Embedded systems 1 und 2

Digitale Verarbeitung von Zahlen

Klasse T-2 // 2018-2019

Inhalt

- **Positive und negative ganze Zahlen**
  - Darstellung und Codierung
  - Umwandlung zwischen verschiedenen Basen
  - Zahlenkreis
  - Additionen, Subtraktionen und Vergleiche

- **Reelle Zahlen**
  - Darstellung und Codierungen
  - Genauigkeit
  - Standard Zahlen

Darstellung der positiven ganzen Zahlen

- Die positiven ganzen Zahlen N werden auch Kardinalzahlen genannt (*cardinal numbers*), logische Zahlen oder Zahlen ohne Vorzeichen (*unsigned number*)

- Sie lassen sich in einer gegebenen Basis b durch eine Folge von Ziffern a_i eingeschlossen zwischen 0 und b-1 darstellen, mit:

$$N = \sum_{i=0}^{n-1} a_i \cdot b^i \qquad \rightarrow\ 0 \le a_i \le b\text{-}1$$

- Durch Konvention wird die Zahl N dargestellt durch:

$$N = a_{n-1}a_{n-2}\ldots a_1 a_0 \qquad \rightarrow 0 \le N \le b^n\text{-}1$$

wobei

a_0	die niederwertige Ziffer,	(*LSD: Least Significant Digit*)
a_{n-1}	die höherwertige Ziffer,	(*MSD: Most Significant Digit*)
n	die Anzahl Ziffern (*Digits*) darstellen	

Codierung der positiven ganzen Zahlen

Dezimale Codierung

$$N = \sum_{i=0}^{n-1} a_i \cdot 10^i \qquad \rightarrow 0 \le a_i \le 9$$

das bedeutet für den Wert 1'386:

$1'386_{10} = 1*10^3 + 3*10^2 + 8*10^1 + 6*10^0$

Binäre Codierung

$$N = \sum_{i=0}^{n-1} a_i \cdot 2^i \qquad \rightarrow 0 \le a_i \le 1$$

das bedeutet für den Wert 1'386:

$1'386_{10} = 101'0110'1010_2 = 1*2^{10} + 0*2^9 + 1*2^8 + 0x2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$

Oktale Codierung

$$N = \sum_{i=0}^{n-1} a_i \cdot 8^i \qquad \rightarrow 0 \le a_i \le 7$$

das bedeutet für den Wert 1'386:

$1'386_{10} = 2'552_8 = 2*8^3 + 5*8^2 + 5*8^1 + 2*8^0$

Codierung der positiven ganzen Zahlen (II)

Hexadezimale Codierung

$$N = \sum_{i=0}^{n-1} a_i \cdot 16^i \qquad \rightarrow 0 \leq a_i \leq 15$$

das bedeutet für den Wert 1'386:

$1'386_{10} = 56A_{16} = 5x16^2 + 6x16^1 + 10x16^0$

Beziehung zwischen den Basen 10, 16, 8 und 2

dezimal	hexadezimal	oktal	binär
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Umrechnungen Basis b → dezimal → Basis b

- **Umrechnung Basis b → dezimal**

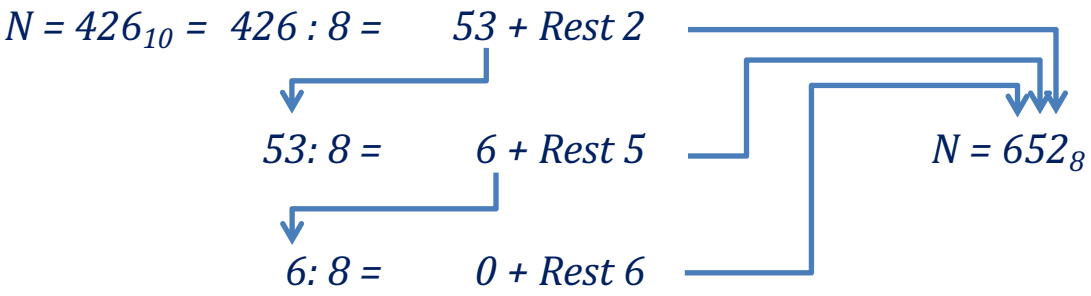
$$N = N_b = a_{n-1}*b^{n-1}+a_{n-2}*b^{n-2}+\ldots a_1*b^1+a_0*b^0 = N_{10}$$

z. B. $N = 652_8 = 6*8^2 + 5*8^1 + 2*8^0 = 426_{10}$

- **Umrechnung dezimal → Basis b**

Die Umrechnung einer Dezimalzahl N in die entsprechende Zahl mit der Basis b kann durch eine Reihe von Divisionen vorgenommen werden.

z. B. Umrechnung der Dezimalzahl 426 in eine Oktalzahl



Umrechnung hexadezimal – binär → hexadezimal

- **Umrechnung hexadezimal → binär**

$N = 8AC9_{16} =$ 8 A C 9 *hexadezimal*

↓ ↓ ↓ ↓

1000 1010 1100 1001 *binär*

$= 1000101011001001_2$

- **Umrechnung binär → hexadezimal**

$N = 1000101011001001_2 =$ $\underbrace{1000}$ $\underbrace{1010}$ $\underbrace{1100}$ $\underbrace{1001}$ *binär*

↓ ↓ ↓ ↓

8 A C 9 *hexadezimal*

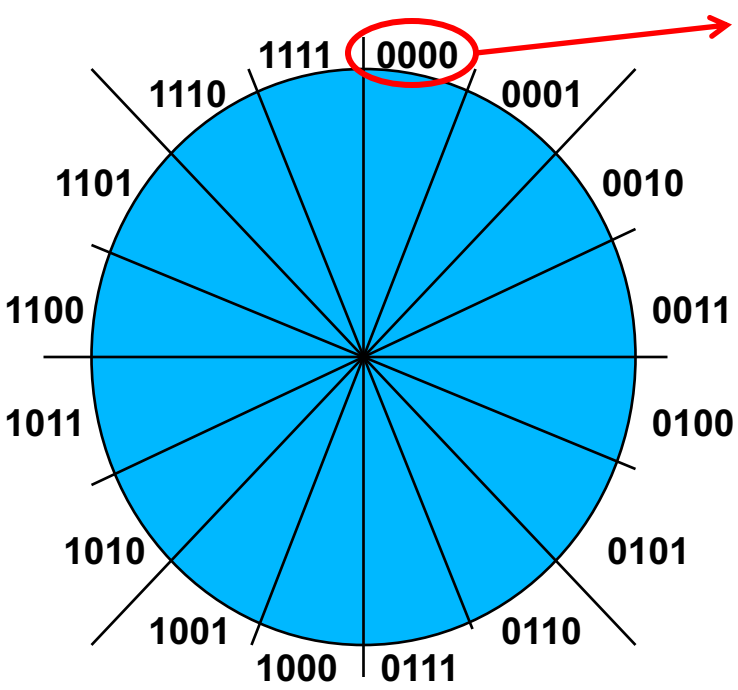
$= 8AC9_{16}$

Zahlenkreis

- **Eine begrenzte Anzahl Bit für die binäre Darstellung eines Wertes (zum Beispiel 8 Bit) wird ihren Darstellungsbereich begrenzen.**

  Mit 8 Bit kann ein Wert zwischen 0 und 255 dargestellt werden → [0 .. 2^n-1]
  oder binär 00000000 und 11111111

- **Darstellung der Werte eines 4-Bit-Registers (n=4) in einem Kreis**



$$Z = \overline{b}_3 * \overline{b}_2 * \overline{b}_1 * \overline{b}_0$$

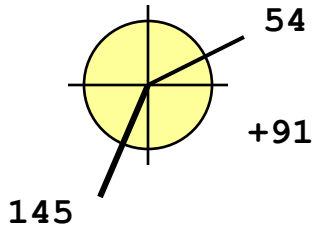
Binäre Addition der Zahlen ohne Vorzeichen

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

```
ldr     r0, =54
ldr     r1, =91
adds    r2, r0, r1
```

```
  54: 00110110
+ 91: 01011011
---------------
 145:010010001
```

Carry (C) = 0

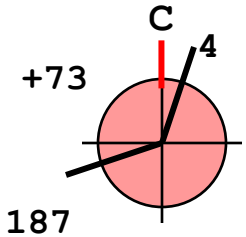


Kapazitätsproblem

```
ldr     r0, =187
ldr     r1, =73
adds    r2, r0, r1
```

```
 187: 10111011
+ 73: 01001001
---------------
   4:100000100
```

Carry (C) = 1



Binäre Subtraktion der Zahlen ohne Vorzeichen

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

```
ldr     r0, =54
ldr     r1, =45
subs    r2, r0, r1
```

```
borge                  10
      54: 00110110
     -45: 00101101
behalte                 1
--------------------
       9:000001001
```

Carry (C) = 1

da das Carry bei einer Subtraktionsoperation mit Zahlen ohne Vorzeichen invertiert wird

Kapazitätsproblem

```
ldr     r0, =127
ldr     r1, =128
subs    r2, r0, r1
```

```
borge        10
     127: 01111111
    -128: 10000000
behalte   1
--------------------
   255:111111111
```

Carry (C) = 0

da das Carry bei einer Subtraktionsoperation mit Zahlen ohne Vorzeichen invertiert wird

Subtraktion mit 1er-Komplement

- Die Addition zweier positiver Zahlen bietet überhaupt kein Problem und lässt sich ganz einfach durchführen, daher die Idee, die Subtraktion zweier Zahlen mit einem Addierer durchzuführen ($V = A - B$).

- Die Methode benutzt einen Versatz R, damit $R - B$ positiv wird.

$$V = A - B = A + (R - B) - R = A + \overline{B} - R$$

- Für eine Binärzahl mit n Bit wählt man:

$$R = 2^n - 1 = \sum_{i=0}^{n-1} 2^i$$

- Das Komplement $\overline{B}$ einer Zahl B ist definiert durch:

$\overline{B} = R - B \quad \rightarrow B = 01010110 \rightarrow \overline{B} = 10101001$ → Bitinversion

- Die Subtraktion zweier Zahlen wird zurückgeführt auf:

$$V = A + \overline{B} - R = A + \overline{B} - (2^n - 1) = A + \overline{B} + 1 - 2^n$$
$$V = A + \overline{B} + 1$$

Subtraktion mit 2er-Komplement

- Die Methode mit dem 2er-Komplement stützt sich auf das gleiche Prinzip wie beim 1er-Komplement, aber über einen Versatz R gleich:

$$R = 2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

- Das 2er-Komplement $\dot{B}$ einer Zahl B ist definiert durch:

$\dot{B} = R - B = \overline{B} + 1$ → $B = 01010110$ → $\dot{B} = 10101010$ → Inversion der Bit + 1

- Die Subtraktion zweier Zahlen wird zurückgeführt auf:

$V = A + \dot{B} - R = A + \dot{B} - 2^n$

$V = A + \dot{B}$

Subtraktion im 2er-Komplement von zwei Zahlen ohne Vorzeichen

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

```
ldr     r0, =54
ldr     r1, =45
subs    r2, r0, r1
```

```
     54: 00110110
+ (-45): 11010011
--------------------
      9:100001001
```

Carry (C) = 1

Kapazitätsproblem

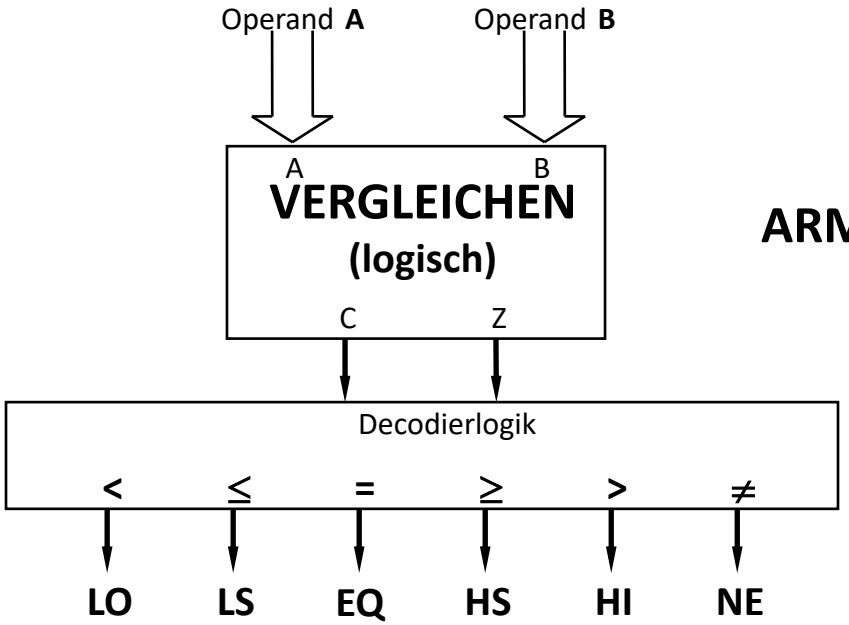
```
ldr     r0, =64
ldr     r1, =100
subs    r2, r0, r1
```

```
      64: 01000000
+ (-100): 10011100
--------------------
    (-36):011011100
```

→ 220!!!

Carry (C) = 0

Vergleich von vorzeichenlosen Zahlen



ARM: CMP <Ra>, <Rb>

Vergleich = A - B

LO:	(Lower)	C == 0	A < B (strikt kleiner)
LS:	(Lower or same)	C == 0 || Z == 1	A ≤ B (kleiner oder gleich
EQ:	(Equal)	Z == 1	A = B (gleich)
NE:	(Not equal)	Z == 0	A ≠ B (ungleich)
HS:	(Higher or same)	C == 1	A ≥ B (grösser oder gleich)
HI:	(Higher)	C == 1 && Z == 0	A > B (strikt grösser)

Darstellung der negativen ganzen Zahlen

- Die ganzen Zahlen N, die sowohl positiv als auch negativ sein können, werden auch arithmetische Zahlen oder Zahlen mit Vorzeichen (*signed numbers*) genannt

- Für Zahlen mit Vorzeichen existieren verschiedene Darstellungsarten
  - Darstellung mit zusätzlicher Binärziffer (Vorzeichen)
  - Darstellung mit Versatz
  - Darstellung mit 1er-Komplement
  - Darstellung mit 2er-Komplement

- **Darstellung mit zusätzlicher Binärziffer (Vorzeichen)**
  - Natürliche Darstellung der negativen Zahlen, die daraus besteht, dem Wert ein Vorzeichenbit voranzustellen, z. B. 0 für eine positive und 1 für eine negative Zahl

    z. B. $V = +125_{10} = 01111101_2$

    $V = -125_{10} = 11111101_2$

  - Ein Format mit n Bit lässt sich wie folgt darstellen

    $-(2^{n-1} - 1) \leq N \leq 2^{n-1} - 1$ mit zwei Darstellungen von Null: $+0$ und -0

- **Darstellung mit Versatz**
  - Eine positive oder negative Zahl N wird als Zahl V wie folgt dargestellt:

    $V = N + R$
  - R ist ein positiver Versatz, der so gewählt wird, dass V immer positiv wird und die Bereiche der positiven und der negative Zahlen ungefähr gleich sind:

    $R = 2^{n-1} - 1$ *oder* $R = 2^{n-1}$
  - Die Notation mit Versatz wird für die Exponenten der Gleitkommazahlen verwendet

Die 1er- und 2er-Komplemente

Das 1er-Komplement

Binäre Codierung	Wert
00000000	(+)0
00000001	1
00000010	2
…	…
01111101	125
01111110	126
01111111	127
10000000	-127
10000001	-126
10000010	-125
…	
11111101	-2
11111110	-1
11111111	(-0)

Das 2er-Komplement

Binäre Codierung	Wert
00000000	(+)0
00000001	1
00000010	2
…	…
01111101	125
01111110	126
01111111	127
10000000	-128
10000001	-127
10000010	-126
…	
11111101	-3
11111110	-2
11111111	-1

```
25: 00011001 -> 11100110 (1er-Kompl.)

ldr     r0, =25
mvn     r0, r0
```

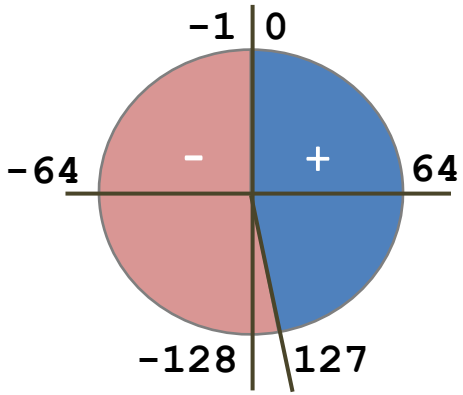
```
25 : 00011001     -> 11100110 (1er-Kompl.)
(1er-Kompl.) + 1  -> 11100111 (2er-Kompl.)

ldr     r0, =25
rsb     r0, r0, #0
```

Negative Flag (N) und die negativen Werte

- 8-Bit-Register (-128 bis 127) → $[-2^{n-1} .. 2^{n-1}-1]$

- Das Flag N (Negative) ist die Kopie des höherwertigen Bit (bit7 von 8 Bit).
  Es wird für Werte von -2^{n-1} bis -1 (-128 bis -1) auf 1 gesetzt (set)



Zum Beispiel: -74 : 10110110

```
ldr     r0, =-74
cmp     r0, #0
```

Flag N = 1

Flag oVerflow (V) und die Addition

- Flag V (oVerflow) erlaubt die Erkennung einer Inkohärenz des Vorzeichens im Ergebnis.
- Dieses Flag wird bei der Verarbeitung von Werten mit Vorzeichen verwendet.

```
ldr     r0, =54
ldr     r1, =10
adds    r2, r0, r1
```

```
  54: 00110110
+ 10: 00001010
--------------
  64: 01000000
```

V=0 → kohärentes Ergebnis

```
ldr     r0, =54
ldr     r1, =91
adds    r2, r0, r1
```

```
  54: 00110110
+ 91: 01011011
--------------
 145: 10010001
```

→ -111!!!

V=1 → inkohärentes Ergebnis

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

Flag oVerflow (V) und die Subtraktion

- **Subtraktion eines positiven Wertes von einem negativen Wert**
  → das Ergebnis müssste immer negativ sein

```
ldr     r0, =-10
ldr     r1, =5
subs    r2, r0, r1
```

```
   -10: 11110110
+ (-5): 11111011
  ---------------
   -15: 11110001
```

V=0 → kohärentes Ergebnis

```
ldr     r0, =-64
ldr     r1, =100
subs    r2, r0, r1
```

```
     -64: 11000000
+ (-100): 10011100
     ---------------
      92: 01011100
```

V=1 → inkohärentes Ergebnis

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

Flag oVerflow (V) und die Spezialfälle

- **In den nachstehenden Fällen wird das Flag V immer zurückgesetzt (clear):**
  - Addition eines positiven und eines negativen Wertes

```
ldr     r0, =10
ldr     r1, =-5
adds    r2, r0, r1
```

```
ldr     r0, =-10
ldr     r1, =12
adds    r2, r0, r1
```

  - Subtraktion zweier positiver Werte

```
ldr     r0, =10
ldr     r1, =12
subs    r2, r0, r1
```

- **Es gibt einen oVerflow, wenn:**
  - Die Summe der beiden positiven Zahlen negativ ist
  - Die Summe der beiden negativen Zahlen positiv ist
  - Die Subtraktion einer positiven Zahl von einer negativen Zahl ein positives Ergebnis ergibt
  - Die Subtraktion einer negativen Zahl von einer positiven Zahl ein negatives Ergebnis ergibt

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

Vollständiges Beispiel

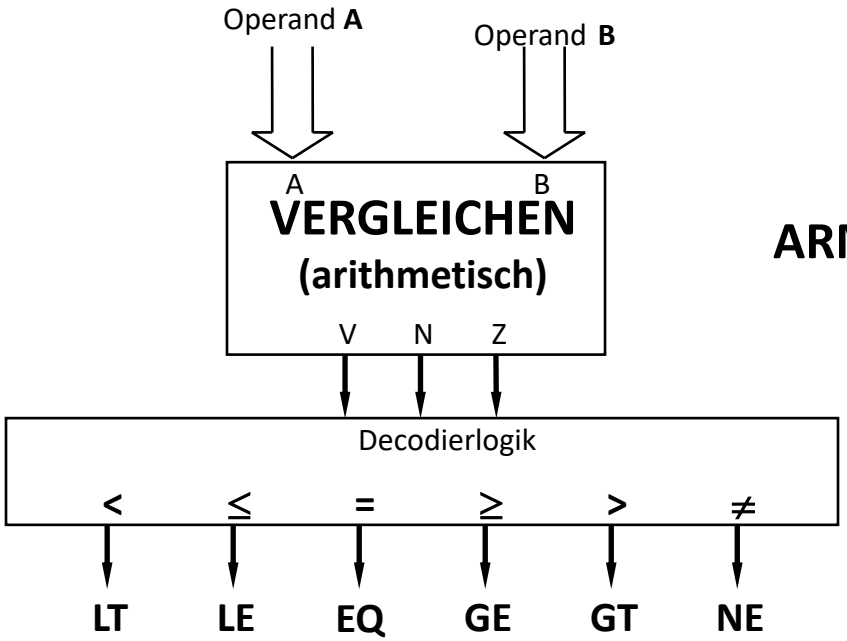
```
ldr     r0, =224        ou -32
ldr     r1, =213        ou -43
adds    r2, r0, r1
```

```
 224: 11100000
 213: 11010101
 ---------------
  110110101   -> 181 oder -75
```

C=1; N=1; V=0; Z=0

Achtung: Aus Gründen der Einfachheit sind die Beispiele in 8-Bit-Worten realisiert, der ARM9 ist aber eine 32-Bit-Maschine und kann diese Tests nur mit 32 Bit durchführen.

Vergleich von Zahlen mit Vorzeichen



ARM: CMP <Ra>, <Rb>

Vergleich = A - B

LT:	(Lower than)	$V \oplus N == 1$	$A < B$ (strikt kleiner)
LE:	(Lower or equal)	$V \oplus N == 1 \,\|\|\, Z == 1$	$A \leq B$ (kleiner oder gleich)
EQ:	(Equal)	$Z == 1$	$A = B$ (gleich)
NE:	(Not equal)	$Z == 0$	$A \neq B$ (ungleich)
GE:	(Greater or equal)	$V \oplus N == 0$	$A \geq B$ (grösser oder gleich)
GT:	(Greater than)	$V \oplus N == 0 \,\&\&\, Z == 0$	$A > B$ (strikt grösser)

Darstellung der reellen Zahlen

- Die reellen Zahlen N können mit einer Basis b durch eine unendliche Folge von Ziffern a_i eingeschlossen zwischen 0 und b-1 dargestellt werden, mit:

$$N = \sum_{i=-\infty}^{n-1} a_i \cdot b^i$$

- einem ganzen Teil N_e und einem gebrochenen Teil N_f, definiert durch:

$$N_e = \sum_{i=0}^{n-1} a_i \cdot b^i \qquad N_f = \sum_{i=-\infty}^{-1} a_i \cdot b^i$$

- Auf die gleiche Weise wie die ganzen Zahlen werden auch die reellen Zahlen durch eine Folge von von links nach rechts nach absteigender Wertigkeit angeordneten Ziffern und einem Komma dargestellt, das den ganzen Teil vom gebrochenen Teil trennt.

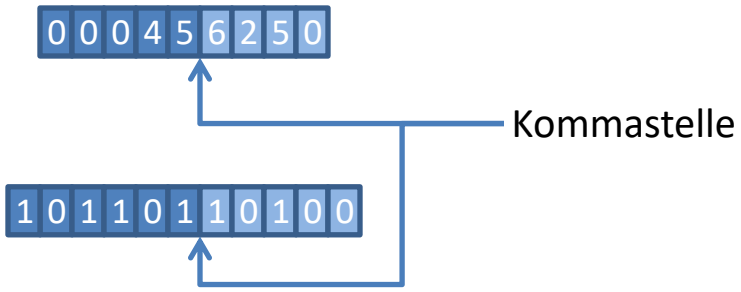
- In der Informatik können zwei verschiedene Darstellungen herangezogen werden.
  - Festkommadarstellung
  - Gleitkommadarstellung

Festkommadarstellung

- In der Festkommadarstellung (*fixed point*) werden die Zahlen als Worte mit einer festen Anzahl Ziffern für den ganzen Teil und den gebrochenen Teil verarbeitet.

z. B. N_{10} = 45.625

N_2 = 10'1101.10100

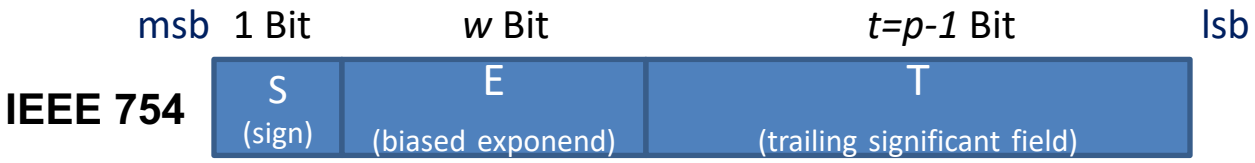


- Übung:

  Binärdarstellung des Wertes 456.3467

Darstellung der Gleitkommazahlen

- **Die reellen Gleitkommazahlen N (*floating point*) werden in der folgenden Form dargestellt:**



$$N = (-1)^S * 2^{E\text{-Versatz}} * 1,T$$

  - S: Vorzeichen des Wertes (0 → "+" und 1 → "-")
  - E: Exponent → Wert + Versatz (hängt von der Grösse des reellen Wertes ab)
  - T: Mantisse → 1, Bruchteil

- **Beispiel:**
  - N = 45.625 → + 101101,101 * 2^0 → 1,0110 1101 * 2^5
  - 32-Bit-Codierung mit einem Versatz von 127 (8 Bit)
    - S = 0
    - E = 5 + 127 = 132 = 0x84
    - T = 01101101

      → 0 100'0010'0 011'0110'1000'0000'0000'0000

      → 0x4236'8000

Einige wichtige Hinweise zur Darstellung der reellen Gleitkommazahlen

- **Normierte Zahlen:**
  - Versatz des Exponenten: $2^{w-1}-1$ — z. B. w=8 → Versatz = 127
  - Bereich des Exponenten: $0 < E < 2^{w}-1$ — z. B. w=8 → $0 < E < 255$
  - Bereich des Bruchs: null und nicht null
- **Unendliche Zahlen:**
  - Vorzeichen: +, -
  - Exponent: $2^{w}-1$ — z. B. w=8 → $E = 255$
  - Mantisse: null
- **NAN (Not a Number):**
  - Vorzeichen: nicht berücksichtigt
  - Exponent: $2^{w}-1$ — z. B. w=8 → $E = 255$
  - Mantisse: nicht null

Genauigkeit der reellen Zahlen

- Bei der Darstellung der reellen Zahlen entsteht ein Genauigkeitsproblem, denn es ist nicht möglich, den gebrochenen Teil mit einer unendlich langen Ziffernfolge darzustellen. Diese ist daher auf u Ziffern begrenzt.

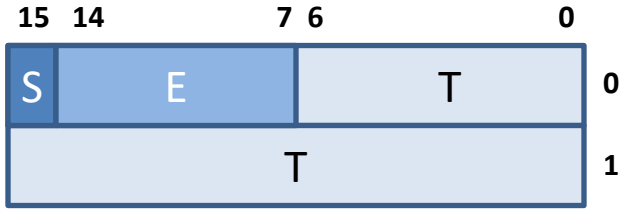
$$N_f \rightarrow \tilde{N}_f = \sum_{i=-u}^{-1} a_i \cdot b^i$$

- Dieses Abschneiden erzeugt einen Fehler kleiner b^{-u}

- In der Praxis wird das Abschneiden durch Runden ersetzt und die Bruchzahl gewählt, die durch Auf- oder Abrunden am nächsten bei N_f liegt. Dies ergibt einen Rundungsfehler, der zwischen $+(b^{-u})/2$ und $-(b^{-u})/2$ liegt

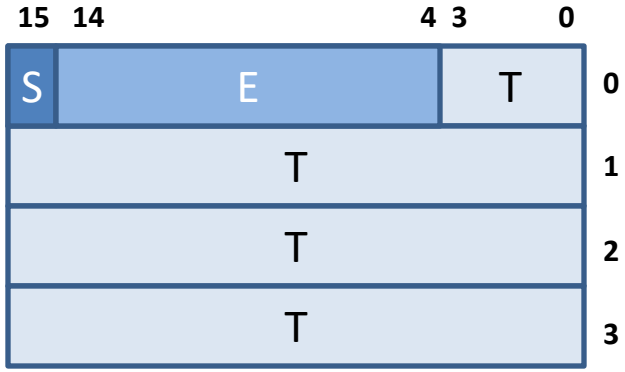
Standard Gleitkommazahlen nach IEEE 754

Format mit einfacher Genauigkeit mit 32 Bit (*float*)



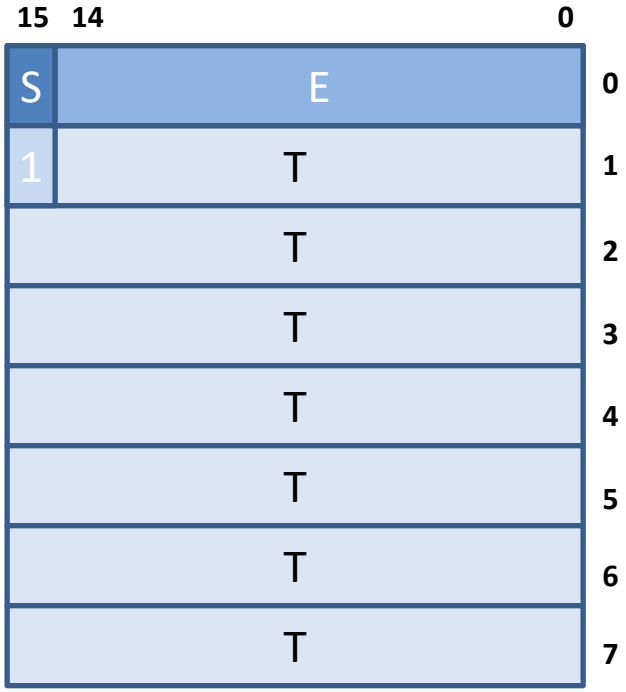
Einfacher Genauigkeit mit 32 Bit
- 1 Vorzeichenbit
- 8 Bit für den Exponenten
- 23 Bit für die Mantisse

Format mit doppelter Genauigkeit mit 64 Bit (*double*)



Doppelte Genauigkeit mit 64 Bit
- 1 Vorzeichenbit
- 11 Bit für den Exponenten
- 52 Bit für die Mantisse

Format mit vierfacher Genauigkeit mit 128 Bit (*long double*)



Vierfache Genauigkeit mit 128 Bit

1	Vorzeichenbit
15	Bit für den Exponenten
1	Bit für den ganzen Teil
111	Bit für die Mantisse