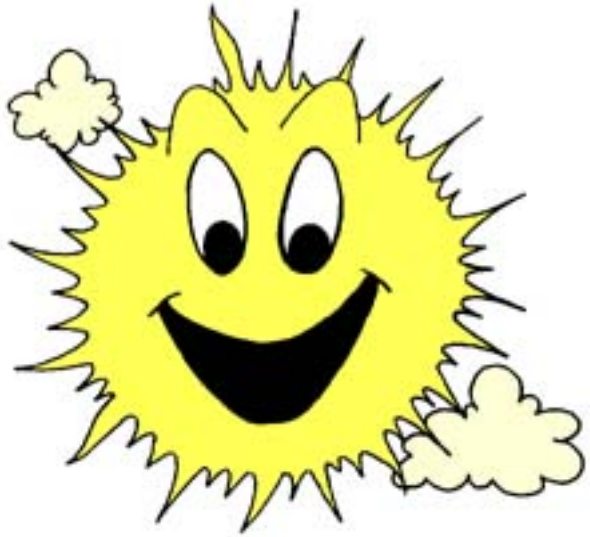
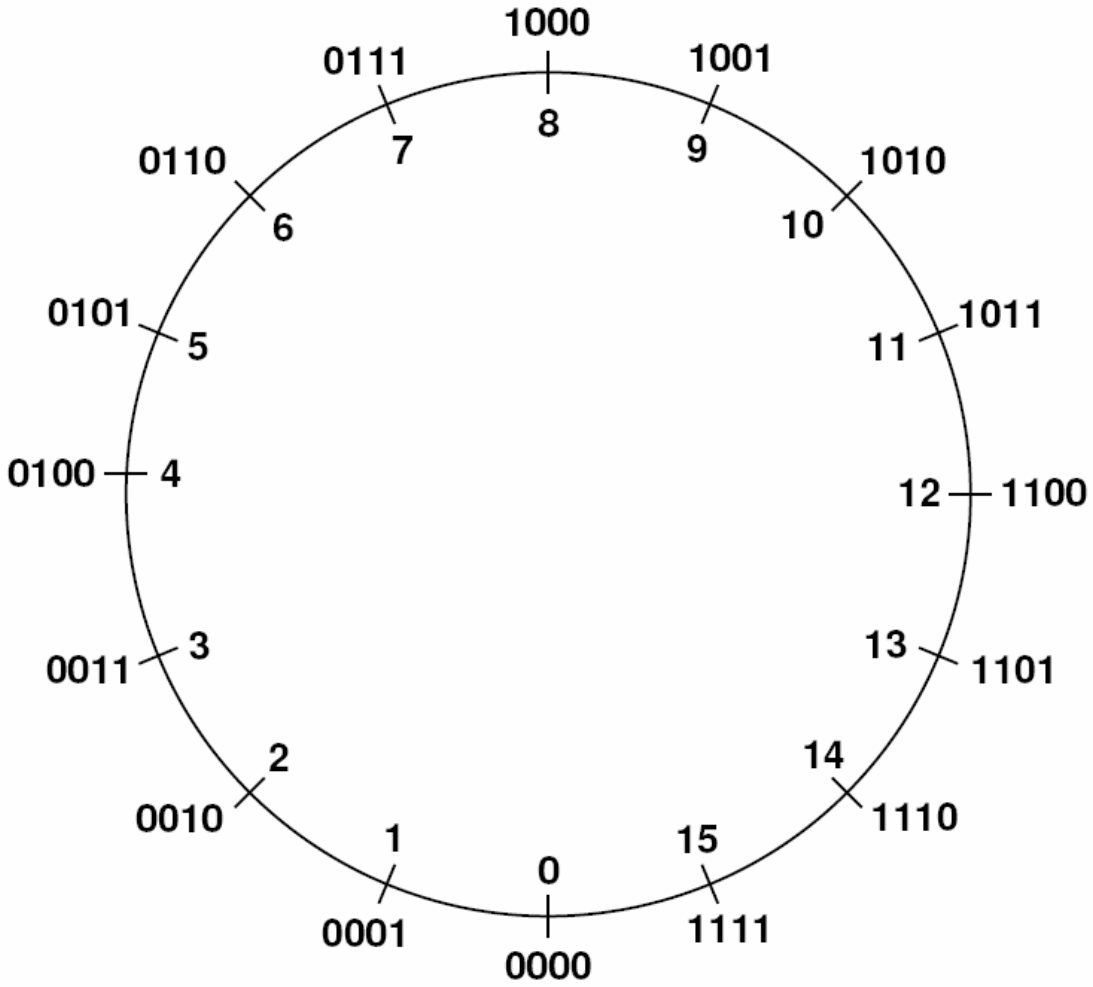
Wiederholung der 2. Vorlesung



- Darstellung von Binärzahlen
  - Ganze Zahlen – Festkomma (positiv und positiv/negativ)
  - 1er- und 2er Komplement
- Übertrag – Carry
- Überlauf – Overflow
- Little und Big Endian.

2.2 Integer Zahlen (ohne Vorzeichen)



$$Z = \sum_i x_i Y^i,$$

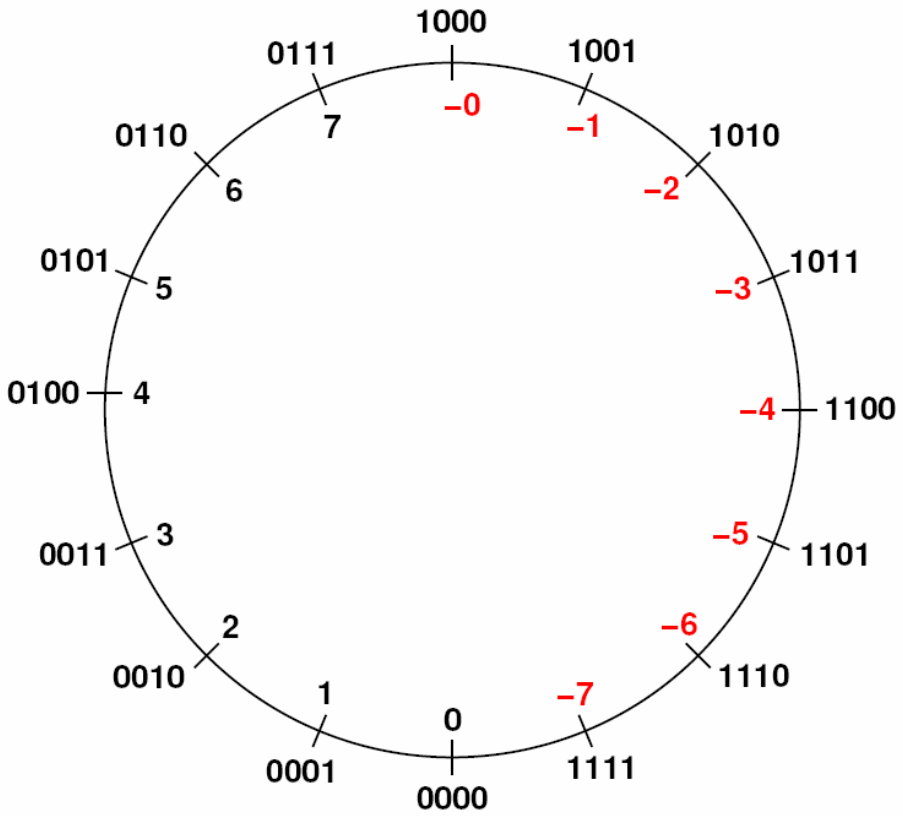
$= x_3*2^3 + x_2*2^2 + x_1*2^1 + x_0*2^0$

$= x_3*8 + x_2*4 + x_1*2 + x_0*1$

Wertebereich:

$0 \leq Z \leq 2^n - 1$

2.3 Integer Zahlen (mit Vorzeichen)



$$Z = (-1)^{x_3} * [x_2*2^2 + x_1*2^1 + x_0*2^0]$$
$$= \pm 1 * [x_2*4 + x_1*2 + x_0*1]$$

Wertebereich:

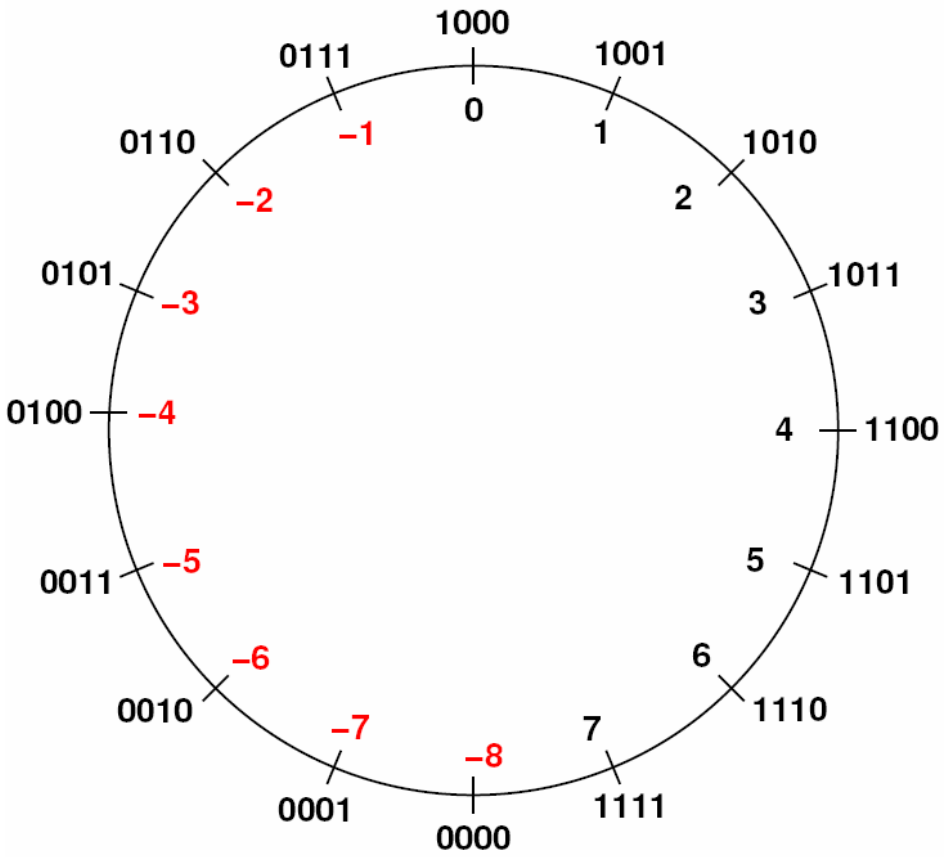
$$-2^{n-1}+1 \leq Z \leq 2^{n-1}-1$$

Most Significat Bit (MSB) definiert das Vorzeichen.

Vorzeichen muss gesondert ausgewertet werden.

Zwei Darstellungen der 0, nämlich +0 und -0

2.4 Integer Zahlen (Binary Offset)



Der kleinst mögliche Wert wird mit 0000... und der größt mögliche Wert wird mit 1111... dargestellt.

Wertebereich:

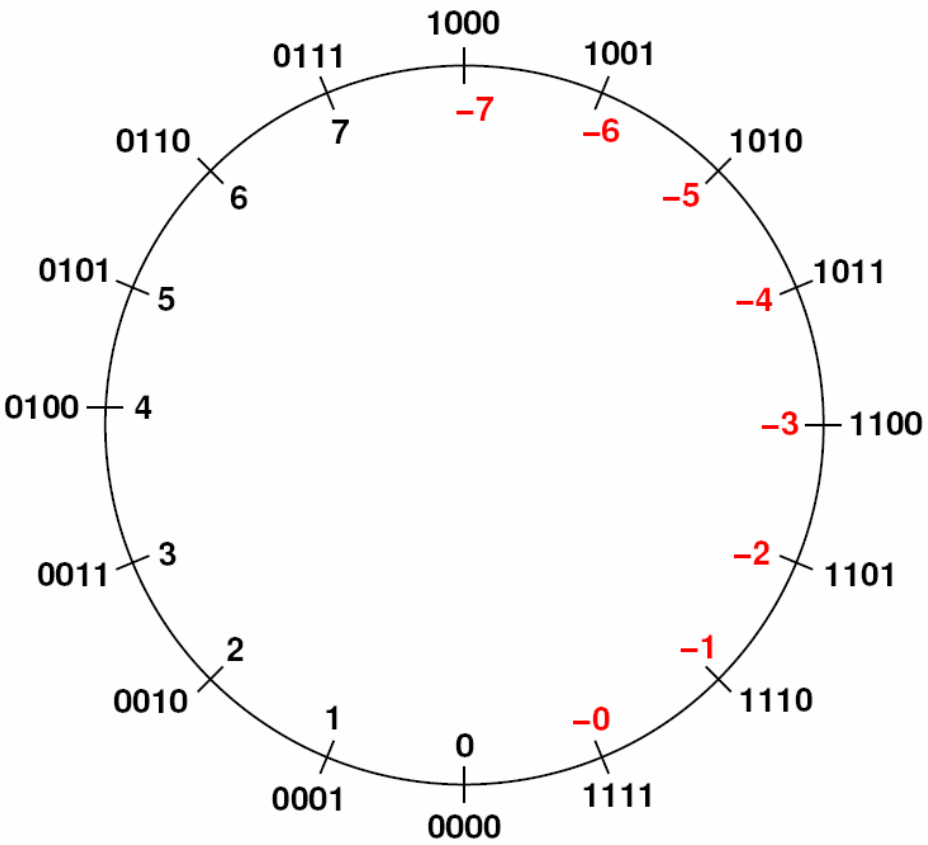
$-2^{n-1} \leq Z \leq 2^{n-1} - 1$

Z = D – O; D: binäre Darstellung, O = 8

Nur eine Darstellung der 0.

Findet Verwendung beim Exponent der IEEE 754

2.5 Integer Zahlen (1er-Komplement)



Most Significant Bit (MSB) definiert das Vorzeichen.

positive Zahl: MSB = 0, (3) niederwertigen Bits werden wie gewohnt ausgewertet

negative Zahl: MSB = 1, (3) niederwertigen Bits werden invertiert und anschließend ausgewertet

Wertebereich:

$$-2^{n-1}+1 \leq Z \leq 2^{n-1}-1$$

2.4 Festkomma-Zahlen (2)

$$(123{,}4567)_{10} = 0\times1000 + 1\times100 + 2\times10 + 3\times1 + \frac{4}{10} + \frac{5}{100} + \frac{6}{1000} + \frac{7}{10000}$$

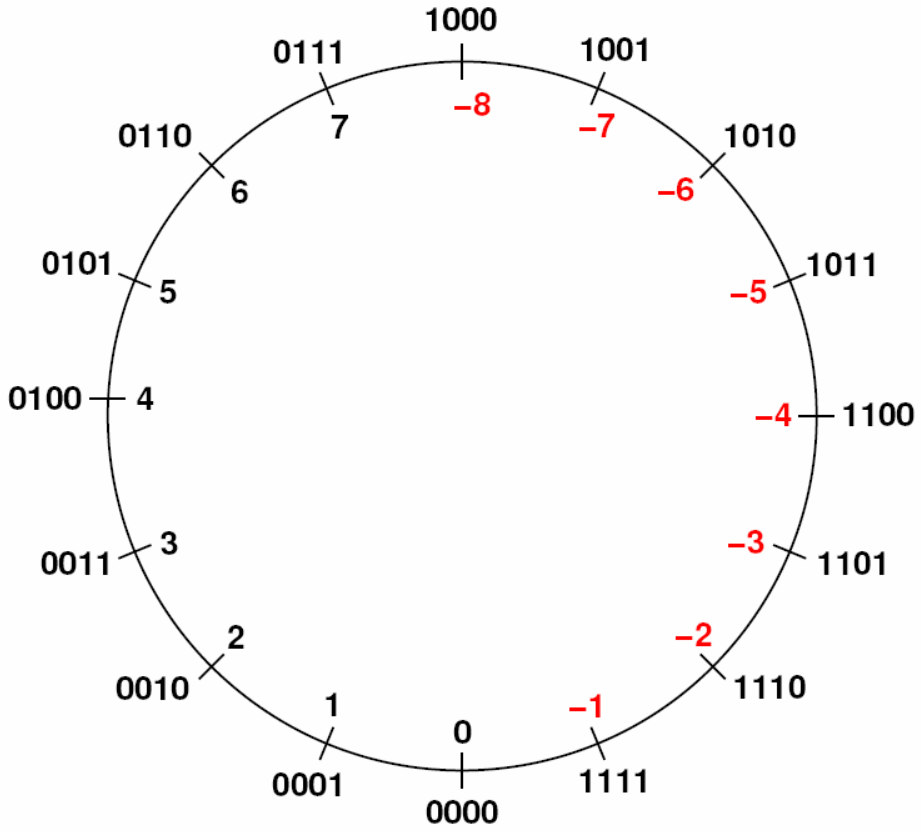
in gleicher Weise kann man binäre Zahlen darstellen:

$$(1001{,}1101)_2 = 1\times8 + 0\times4 + 0\times2 + 1\times1 + \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}$$

In diesem Beispiel ist also $(1001{,}1101)_2 = (9{,}8125)_{10}$

→ Bei Festkomma-Zahlen ist der Wertebereich stark eingeschränkt!

2.5 Integer Zahlen im 2er-Komplement



Most Significant Bit (MSB) definiert das Vorzeichen.

positive Zahl: MSB = 0, (3) niederwertigen Bits werden wie gewohnt ausgewertet

negative Zahl: MSB = 1, 3 niederwertigen Bits werden invertiert, **1 addiert** und anschließend ausgewertet

Wertebereich:

$$-2^{n-1} \leq Z \leq 2^{n-1} - 1$$

2.5 Beispiel 2er-Komplement Festkomma

Informatik

Umwandlung **binär → dezimal**, MSB = 1

$a = (1011,0110)_2$

- falls MSB = 1 → Zahl ist negativ, dann
- Betrag berechnen
  - Komplement aller Bits, 1 addieren
  - → $(1011,0110)_{2\text{-}2K} = (0100,1001)_2 + (1)_2$
    $= (0100,1010)_2$
  - $|a| = (4,625)_{10}$
- Vorzeichen negativ
- **a = -4,625**

2.5 Beisp. 2er-Komplement Festkomma (2)

Informatik

Umwandlung **dezimal → binär**, a < 0

$a = (-4{,}625)_{10}$

- falls a < 0 → Zahl ist negativ, dann
- Betrag berechnen:

Vorkomma	*Nachkomma*
4:2 = 2, R=0	**0,625* 2 = 1,25**
2:2 = 1, R=0	**0,25 * 2 = 0,5**
1:2 = 0, R=1	**0,5 * 2 = 1,0**

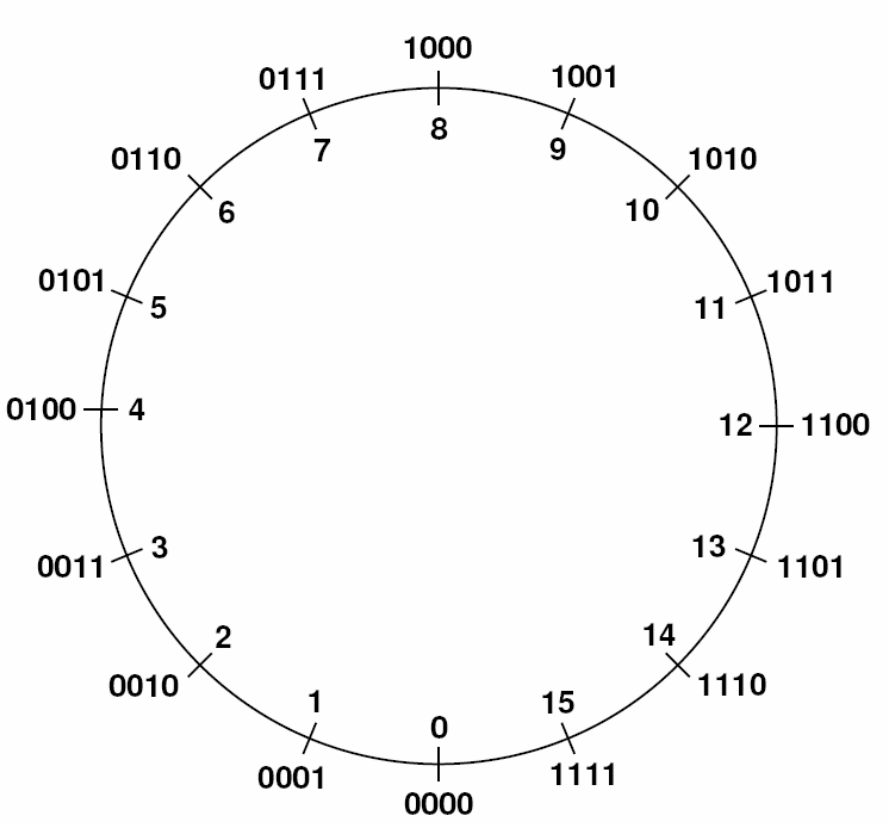
→ |a| = 0100,1010

  - Komplement aller Bits, 1 addieren
  - → $(0100{,}1010)_{2\text{-}2K} = (1011{,}0101)_2 + (1)_2$
    $= (1011{,}0110)_2$

- **$a = (1011{,}0110)_2$**

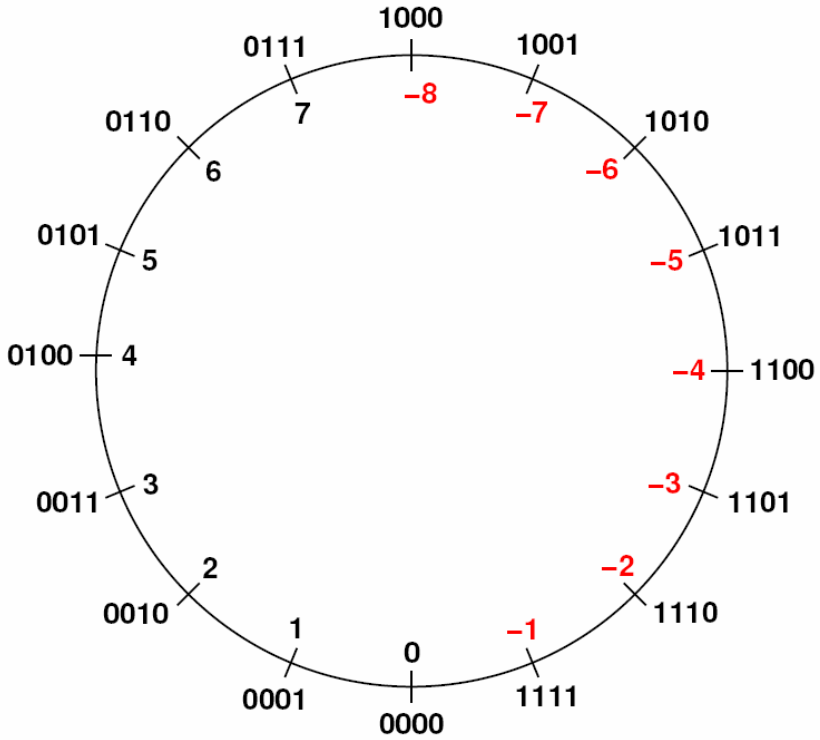
2.6 Übertrag (Carry)

- Ein Übertrag entsteht, wenn bei Operationen mit nur positiven Zahlen der Wertebereich überschritten wird.



2.6 Überlauf (Overflow)

- Ein Überlauf entsteht, wenn bei Operationen von 2er-Komplement-Zahlen der Wertebereich überschritten wird.
- Ein Überlauf liegt dann vor, wenn beide Summanden ein gleiches Vorzeichen haben und das Ergebnis ein anderes.



2.7 Little Endian – Big Endian

Bei der Interpretation, z.B. einer Integer-Zahl, ist zu unterscheiden, ob das niedrigstwertige Byte an der höchsten Adresse („big endian“) oder der niedrigsten Adresse („little endian“) gespeichert wird.

Beispielsweise Speicherung eines 4 Byte Langwort:

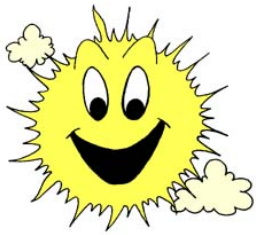
MSB → Byte3 Byte2 Byte1 Byte0 ← LSB

Speicheradresse	Little Endian	Big Endian
BasisAddresse+0	Byte0	Byte3
BasisAddresse+1	Byte1	Byte2
BasisAddresse+2	Byte2	Byte1
BasisAddresse+3	Byte3	Byte0

Little Endian: PC (Intel, AMD), DEC, ...

Big Endian: Apple, IBM370

Ende der Wiederholung



2.2 Binär-kodierte Dezimalzahlen

Bei Taschenrechnern und früheren Tischrechnern blieb man wegen der Ungenauigkeit, die sich bei der Umwandlung in eine andere Basis ergibt, bei der Rechnung mit dezimalen Zahlen.

→ binary coded decimals (BCD)

2.2 Binär-kodierte Dezimalzahlen (2)

- Binary Coded Decimal (BCD)
- jede Dezimalziffer wird mit 4 Bit dargestellt
- jede Einheit mit vier binären Stellen wird als Tetrade (Vierergruppe) bezeichnet
- von insgesamt 16 möglichen Tetraden werden nur 10 genutzt
- die nicht genutzten werden Pseudo-Tetraden genannt

2.2 Binär-kodierte Dezimalzahlen, BCD (3)

Dezimal	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Tetraden

Wertebereich
$0 \leq a \leq 9$

Pseudotetraden

2.2 Beispiel Binär-kodierte Dezimalzahlen

0 0 1 0	0 1 1 0	1 0 0 1
2	6	9

=269

2.2 3-Exzess Code, Aiken Code

- Beim 3-Exzess Code werden die ersten und letzten drei Tetraden der 16 möglichen nicht verwendet. Die Tetrade 0000 kommt nicht vor. Das Neuner-Komplement kann durch einfache Inversion gebildet werden.
- Beim Aiken Code werden die ersten und die letzten 5 Tetraden verwendet. Das Neuner-Komplement kann durch einfache Inversion bestimmt werden.

(Tetrade: ←griechisch, Vierergruppe)

2.2 3-Exzess Code

Dezimal	D	C	B	A
	0	0	0	0
	0	0	0	1
	0	0	1	0
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Pseudotetraden

Symmetrie

Neuner-komplement K_9 durch Inversion

Pseudotetraden

2.2 Gray Code, erweiterter Gray Code

Dezimal	G	R	A	Y
0	0	0	0	0
1	0	0	0	**1**
2	0	0	**1**	1
3	0	0	1	**0**
4	0	**1**	1	0
5	0	1	1	**1**
6	0	1	**0**	1
7	0	1	0	**0**
8	**1**	1	0	0
9	1	1	0	**1**
10	1	1	**1**	1
11	1	1	1	**0**
12	1	**0**	1	0
13	1	0	1	**1**
14	1	0	**0**	1
15	1	0	0	**0**

Beim Gray Code ändert sich beim Übergang von einer Tetrade auf die nächste immer nur ein Bit.

→einschrittiger Code

Gilt nicht bei Übergang von $9_{(10)}$ auf $0_{(10)}$

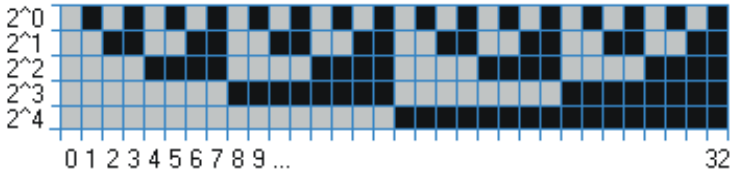
→ Code nicht zyklisch

Erweiterter Gray Code aber ist zyklisch von $15_{(10)}$ auf $0_{(10)}$ weil einschrittig!

2.2 Beispiel Gray Code

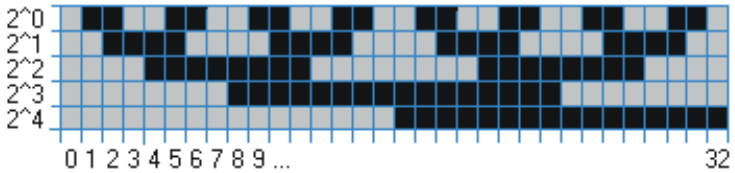
Abtastung von Winkeln oder Positionen durch optische oder mechanische Sensoren:

Dual-Code (mehrschrittiger Code)



Probleme/Fehler durch mechanische Toleranzen

Gray-Code (einschrittiger Code)



Einschrittiger, zyklischer Code hat Vorteile

Beispiel Winkelgeber

2.2 Zusammenfassung Integer-Zahlen

- Unmittelbare Interpretierbarkeit der Darstellung ist für die Verwendung in Rechnern von geringer Bedeutung.
- Zentrale Bedeutung haben die Vereinfachung von arithmetischen Operationen und eine weitgehende Absicherung gegen Fehler wie Überlauf.
- Optimale Darstellung hängt vom konkreten Einsatz und den Anforderungen ab.

2. Wiederholung Umrechnung in Basis 2

$(235)_{10} = (z)_2$

235 : 2 = 117,	Rest 1	$2^0 * 1$	1 +
117 : 2 = 58,	Rest 1	$2^1 * 1$	2 +
58 : 2 = 29,	Rest 0	$2^2 * 0$	4 +
29 : 2 = 14,	Rest 1	$2^3 * 1$	8 +
14 : 2 = 7,	Rest 0	$2^4 * 0$	
7 : 2 = 3,	Rest 1	$2^5 * 1$	32 +
3 : 2 = 1,	Rest 1	$2^6 * 1$	64 +
1 : 2 = 0,	Rest 1	$2^7 * 1$	128

11101011 = $(235)_{10}$

2. Regel zur Umrechnung einer Dezimalzahl

$$a / b_2 = d_0 + r_0 / b_2$$
$$d_0 / b_2 = d_1 + r_1 / b_2$$
$$\ldots$$
$$d_{n-2} / b_2 = d_{n-1} + r_{n-1} / b_2$$
$$d_{n-1} / b_2 = 0 + r_n / b_2$$

Aus letztem Schritt folgt:

$d_{n-1} = r_n$, daraus folgt:

$$d_{n-2} = r_n \times b_2 + r_{n-1}$$

$$d_{n-1} = r_n$$
$$d_{n-2} = r_n \times b_2^{\,1} + r_{n-1}$$
$$\ldots$$
$$d_1 = r_n \times b_2^{\,n-2} + r_{n-1} \times b_2^{\,n-3} + \ldots + r_2$$
$$d_0 = r_n \times b_2^{\,n-1} + r_{n-1} \times b_2^{\,n-2} + \ldots + r_2 \times b_2^{\,1} + r_1$$
$$a = r_n \times b_2^{\,n} + r_{n-1} \times b_2^{\,n-1} + \ldots + r_2 \times b_2^{\,2} + r_1 \times b_2^{\,1} + r_0 = \sum_{i=0}^{n} r_i \times b_2^{\,i}$$

2. Horner-Schema

Das Hornerschema ist eine effiziente Methode zur Umwandlung von Zahlen zur Basis *b* in das Dezimalsystem.

$$(r_n r_{n-1} \ldots r_1 r_0)_b = (\ldots((r_n b + r_{n-1})b + r_{n-2})b + \ldots + r_1)b + r_0$$

Beispiel:

$1021_3 = ((1x3 + 0)x3 + 2)x3 + 1$

$= 34_{10}$

Schnelle Berechnung nur mit Addition und Multiplikation:
Kein Potenzieren notwendig → schneller, genauer

2.4 Umwandlung von Nachkommastellen

Regel:

$$a \times b = \upsilon_0, a_0$$

$$a_0 \times b = \upsilon_1, a_1$$

$$a_1 \times b = \upsilon_2, a_1$$

...

$$a_{n-1} \times b = \upsilon_n, a_n$$

$$a_{(10)} = (0, \upsilon_0 \upsilon_1 ... \upsilon_{n-1} \upsilon_n)_{(b)}$$

Übung $(0{,}28125)_{10}$

```
0,28125*2 = 0,5625
0,5625*2  = 1,125
0,125*2   = 0,25
0,25*2    = 0,5
0,5*2     = 1,0
```

$(0{,}28125)_{10} = (0{,}\overrightarrow{01001})_2$

2.8 Darstellung einer Fließkomma-Zahl

- Allgemeine Darstellung:
  $z = (-1)^s \times 1.f \times 2^{e-o}$
- s = Vorzeichen (Signum, sign, 0=pos., 1=neg.)
- f = Fraktion, fraction
- e = Exponent
- o = Offset (127 bzw. 1023) wg. Wertebereich ±

1.f: normalisierte Darstellung: 1 vor Binärpunkt ist implizit, d.h. wird angenommen, nicht dargestellt

0.f: denormalisierte Darstellung: in besonderen Fällen wird keine binäre 1 angenommen.

2.8 Darstellung einer Fließkomma-Zahl (2)

Beispiel: Darstellung von $\mathbf{45.625_{10}}$

Vorgehensweise:

1. binäre Darstellung der Vorkommastelle
2. binäre Darstellung der Nachkommastellen
3. → binäre Darstellung der Zahl
4. Normalisierung und Bestimmung Exponent
5. Darstellung von Fraktion f
6. Darstellung des Exponenten e
7. Bestimmung des Vorzeichens s

2.8 Beispiel Fließkomma-Zahl

1. $45_{10} = 101101_2$
2. Nachkommastellen
   $0.625 \times 2 = 1.25$
   $0.25 \times 2 = 0.5$
   $0.5 \times 2 = 1.0$

   $0.625_{10} = 101_2 \times 2^{-3}$
   $= 0.101_2$
3. $45.625_{10} = 45 + 0.625$
   $= 101101.101_2$
4. $101101.101_2 =$
   $1.01101101_2 \times 2^{+5}$
   normalisiert, E = 5
5. Bestimmung Fraktion
   1.01101101, also
   f = 01101101
6. dargestellter Exponent e:
   E = e-o
   e = E+o; o=127 (single)
   = 5+127
   = 132
   $= 10000100_2$
7. Vorzeichen s = 0 (pos)

2.8 Beispiel Fließkomma-Zahl (2)

Beispiel: $\mathbf{45.625_{10}}$

- Fraktion f = 01101101 (beachte 1.01101101)
  bit 0-22
- dargestellter Exponent e = 10000100
  bit 23-30
- Vorzeichen s = 0
  bit 31

Damit ist die Darstellung im IEEE 754-Format mit einfacher Genauigkeit (single precision):

0100 0010 0011 0110 1000 0000 0000 0000

2.8 Spezielle Werte bei IEEE 754

- Zahl **Null** kann nicht direkt dargestellt werden.
  Darstellung durch e=0 und f=0
- Unendlich, **Infinity**
  Darstellung alle Bits im Exponenten 1, f=0;
  Unterscheidung von ±Infinity durch s=0 oder 1
- Not-a-Number **NaN**, Darstellung ist keine Zahl.
  Darstellung alle Bits im Exponenten 1, f≠0;
- Beispiele:

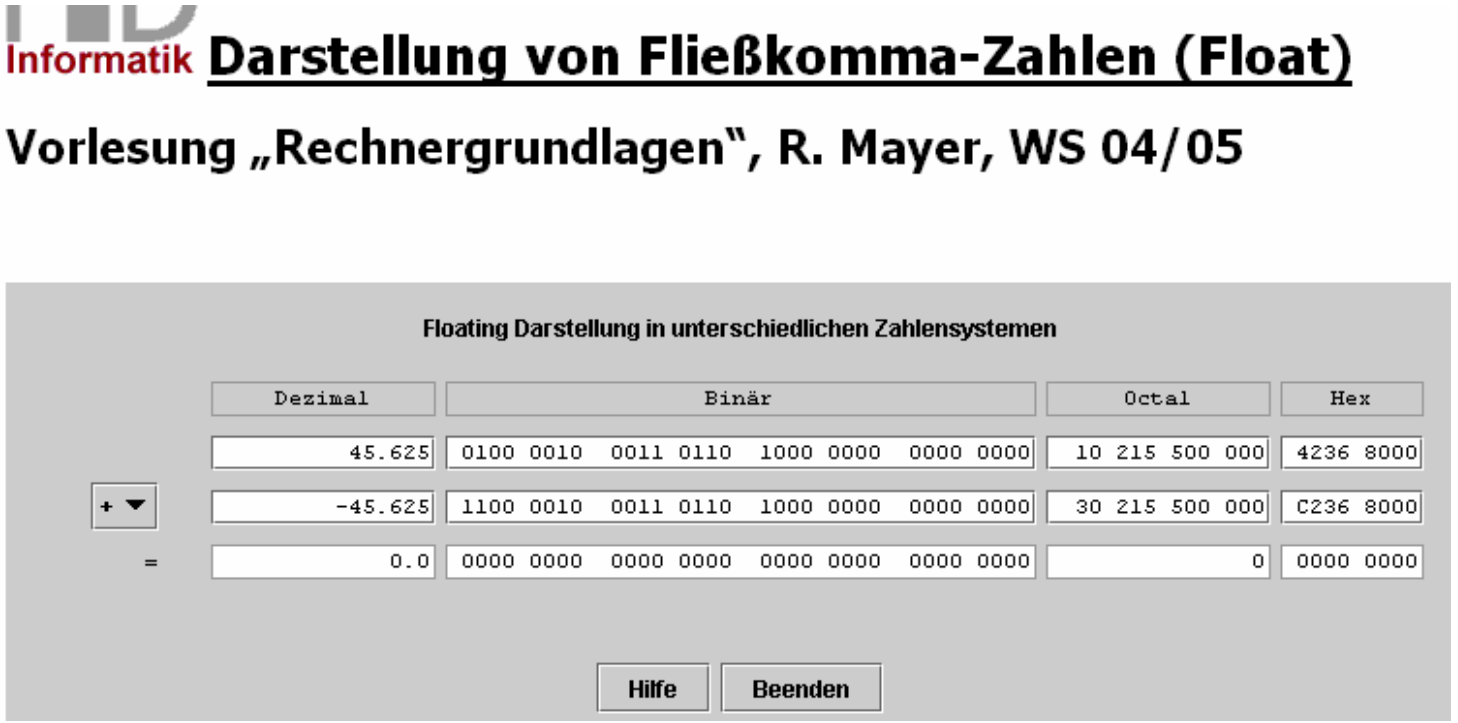
  ±nichtnull / 0 = ±Infinity

  ±Infinity / ±Infinity = NaN u.s.w

2.5 Beispiel Applet: IEEE 754 Darstellung

Beispiel für binäre Darstellung von 32-Bit Floating-Point sowie den vier Grundrechenarten:

http://www.fbi.fh-darmstadt.de/~rmayer/RG/ra/FloatDarst.html



2.8 IEEE 754 (Zusammenfassung)

- Genauigkeit: einfach 32 bit, doppelt 64 bit
- Vorzeichenbit s: 0 positiv, 1 negativ
- Mantisse = 1.f; f: Bits der Fraktion
- Exponent
  - Basis 2
  - Exponent = Exponentendarstellung - Offset
  - -127 und +128 sind reserviert (-1023 und +1024)
  - Falls Exponent == 0 ist Darstellung denormalisiert
    $(-1)^{s} \times 0.f \times 2^{-(\text{Bias}-1)}$; Bias = 127 (einfach), 1023 (doppelt)
- Darstellung und Behandlung der Werte
  - **+Infinity** und **–Infinity** (unendlich)
  - **NaN** (keine darstellbare Zahl)