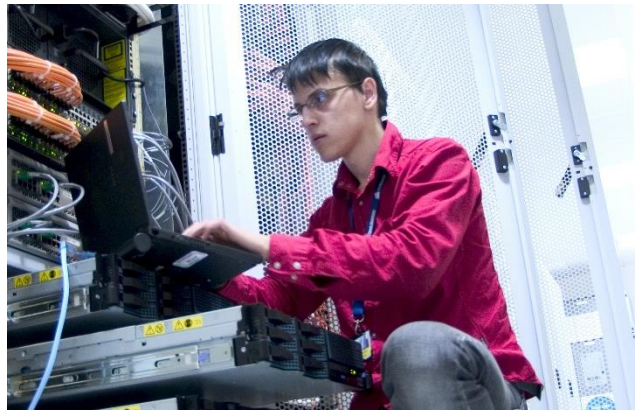
EMIL-VON-BEHRING GYMNASIUM

Darstellung binärer Zahlen



Stellenwertsystem

- Zahlen werden über das Stellenwertsystem dargestellt, bei dem die Wertigkeit eines Symbols/Ziffer von ihrer Position abhängt:
- Beispiel: 127
  - Die Ziffer 1 ist kleiner als die 2 aber an der dritten Stelle von rechts stellt die 1 die Ziffer 1 den Wert 100 dar und die 2 an der zweiten Stelle von rechts nur den Wert 20.
- Hinweis: Das Vorgehen zum schriftlichen Addieren, Subtrahieren, Multiplizieren und Dividieren aus der Grundschule ist auf alle Zahlen anwendbar, die mit dem Stellenwertsystem dargestellt sind.
- Es gibt auch Zahlendarstellungen, die nicht durch das Stellenwertsystem dargestellt sind: z.B. römische Zahlen MCMLXXXIV = 1984

Stellenwertsystem

- Der Wert einer Zahl wird im Stellenwertsystem folgendermaßen berechnet: $\ldots + \boldsymbol{a_3 \cdot b^3 + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0 = \sum_{i=0}^{n} a_i \cdot b^i}$ wobei a die einzelnen Ziffern einer Zahl darstellen und b die Basis/Ziffernvorrat für eine Stelle vorgibt.
  - Dezimalsystem: b = 10, also 10 verschiedene Ziffern 0,…,9
  - Dualsystem/Binärzahlen: b = 2, also 2 verschiedene Ziffern 0 und 1
  - Hexadezimalsystem: b = 16, also 16 verschiedene Ziffern von 0,…,9,a,…,f
    Da wir aber kein einzelnes Zeichen für die Ziffern über 10 kennen, bedient man sich hier der Buchstaben: a=10, b=11, c=12, d=13, e=14, f=15 (usw. bei noch höheren Basen)

- $4g2_{19} = 4 \cdot 19^2 + 16 \cdot 19^1 + 2 \cdot 19^0 = 1444 + 304 + 2 = 1750_{10}$

$1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9_{10}$

Aufgabe

- Gib die gegebenen Zahlen im Dezimalsystem an:
- 111001_2
- 2342_8
- $1abc_{13}$
- inf_{72}
- 111100001111_2
- $9b6f_{16}$

Rückrechnung

- Dualzahlen -> Dezimalzahl

  Zahlenwert im Dezimalsystem $= \sum_{i=0}^{n} a_i \cdot 2^i$

  $1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{10}$

- Dezimalzahl -> Dualzahlen

  $13 \div 2 = 6 \quad Rest\ 1$ (niederwertigste Stelle)

  $6 \div 2 = 3 \quad Rest\ 0$

  $3 \div 2 = 1 \quad Rest\ 1$

  $1 \div 2 = 0 \quad Rest\ 1$ (Hochwertigste Stelle)

  $1750 \div 19 = 92 \quad Rest\ 2$ (niederwertigste Stelle)

  $92 \div 19 = 4 \quad Rest\ 16 \triangleq g$

  $4 \div 19 = 0 \quad Rest\ 4$ (Hochwertigste Stelle)

Aufgabe

- Gib die gegebenen Zahlen im angegeben System an:
- $224_{10} = ?_{2}$
- $124_{10} = ?_{13}$
- $1600_{10} = ?_{8}$
- $65536_{10} = ?_{16}$
- $100_{10} = ?_{2}$
- $786_{10} = ?_{16}$

Binärzahlen im Rechner

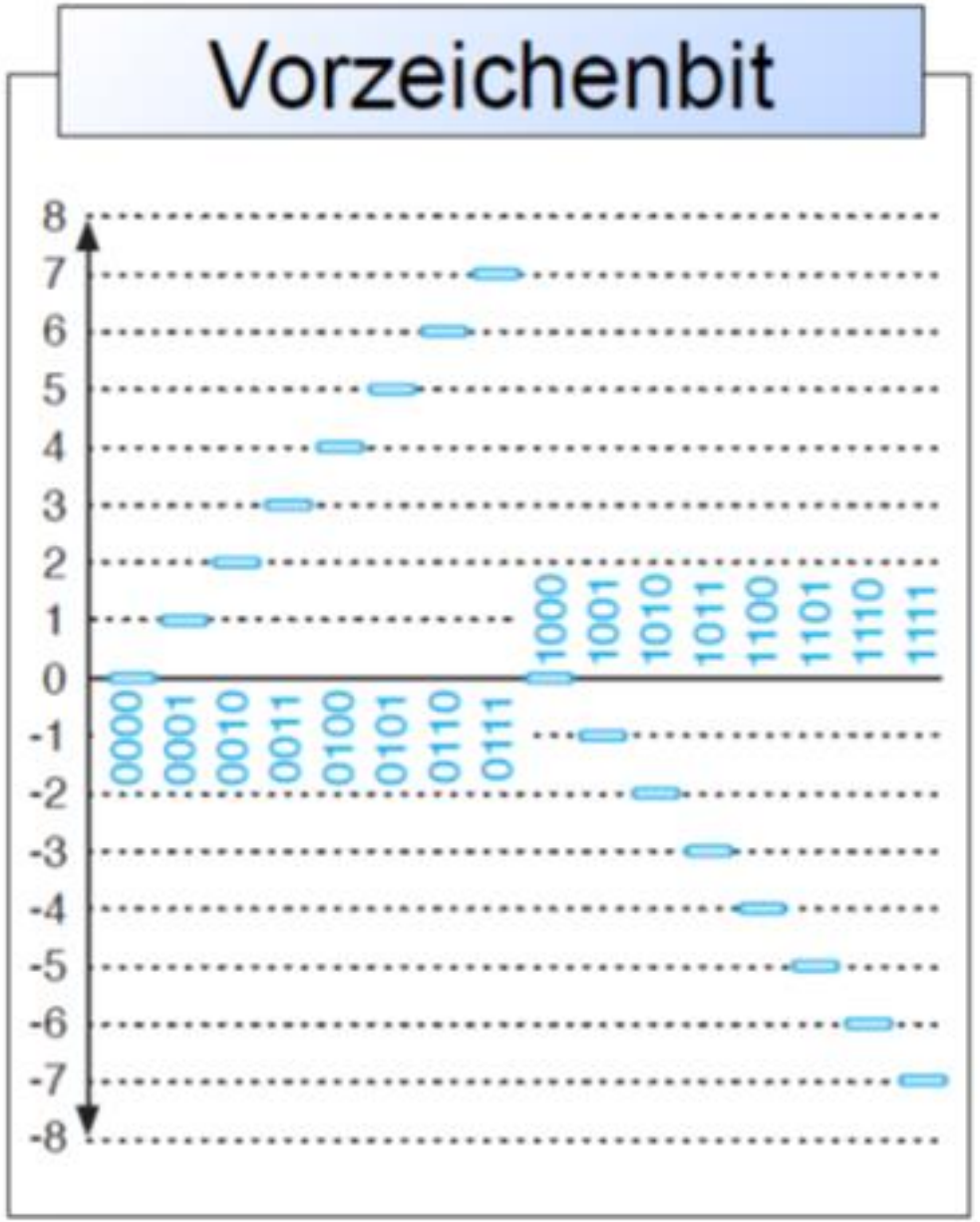
- Jede Zahl kann im Rechner in Binärdarstellung gespeichert werden. Jetzt ist aber die Frage, wie viel Speicherplatz für jede Zahl reserviert werden bzw. wie viel Speicherplatz muss im Speicher eingelesen werden, bis die Zahl vollständig gelesen wurde?
- Daher gibt es Datentypen (auch bei Objekten), die genau vorgeben wie viel Speicherplatz für diese Variable/Objekt reserviert werden muss.
  - Beispiel: Für ein Integer (int) werden 32 Bit (=4 Byte) reserviert.
  - Somit könnten bei diesen begrenzten Speicherplatz die Zahlen 0 (alle Bits sind 0) bis 4.294.967.295 (alle Bits sind 1) angegeben dargestellt werden.
- Aber wie werden jetzt negative Zahlen dargestellt?

Negative Zahlen

- Man benötigt für die Information positiv oder negativ genau ein 1 Bit, weil es genau zwei „Zustände“ gibt, positiv oder negativ. Das erste Bit soll nun angeben, ob die Zahl positiv oder negativ ist. Ist das erste Bit 0, dann ist die Zahl positiv. Ist das erste Bit 1, dann ist die Zahl negativ.
- Hieraus ergeben sind drei/mehrere Varianten wie man die negativen Zahlen kodieren kann:
  - Vorzeichenbit
    Vorzeichen wird durch ein einzelnes Bit definiert
  - Einerkomplement
    Negative Zahlen werden durch das Invertieren aller Bits gebildet
  - Zweierkomplement
    Negative Zahlen: Invertieren und Addieren von 1

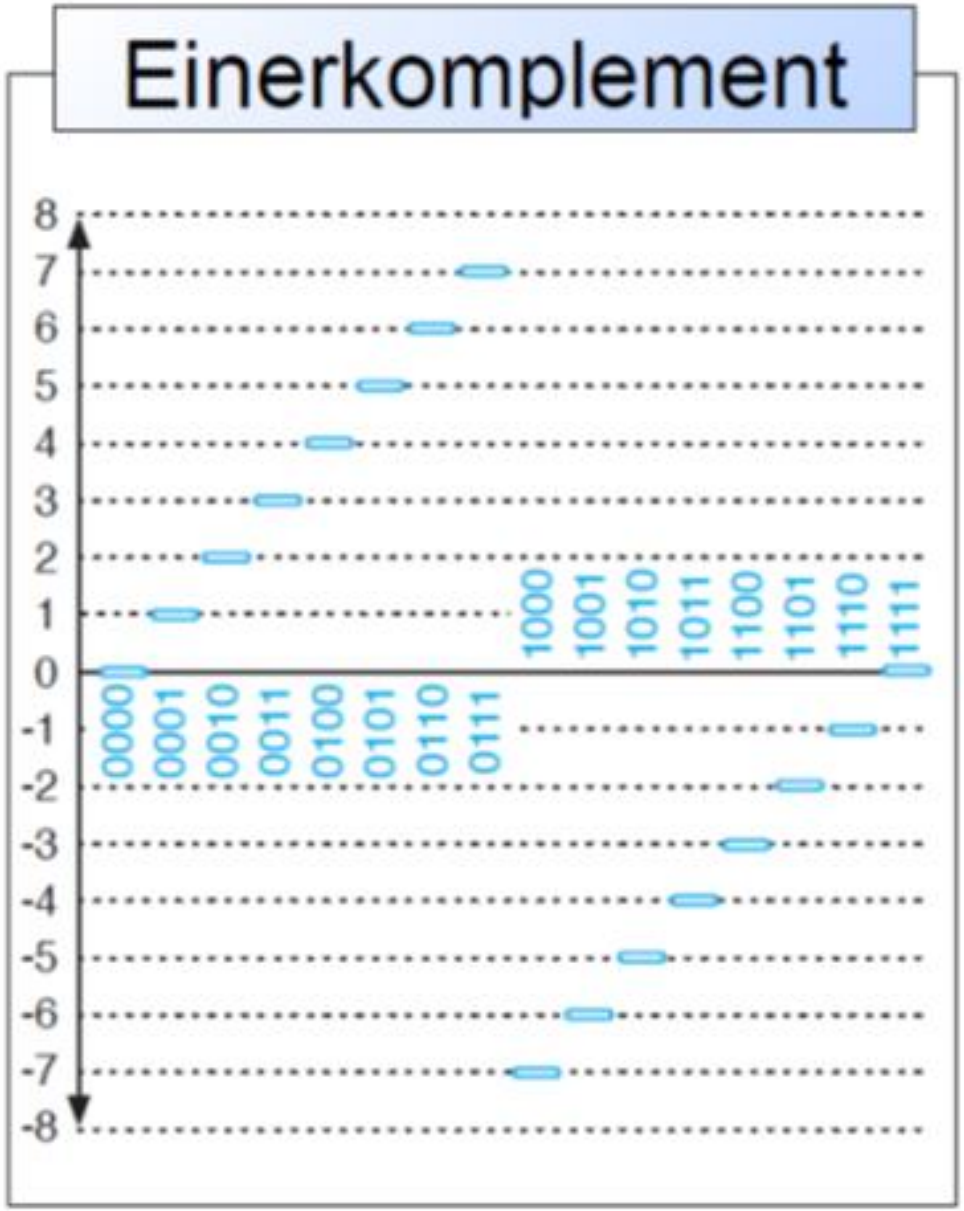
Vorzeichenbit

- Das erste Bit gibt das Vorzeichen an, die restlichen Bits geben die Zahl (Betrag der Zahl) an.
- Beispiele anhand eines Datentyps, der immer 4 Bit im Speicher reserviert:
  - 5=0101 -> -5=1101
  - 3=0011 -> -3=1011
  - 0=0000 -> -0=1000



Einerkomplement

- Das erste Bit gibt auch hier das Vorzeichen an, aber negative Zahlen werden durch das Invertieren aller Bits gebildet.
- Beispiele:
  - 5=0101 -> -5=1010
  - 3=0011 -> -3=1100
  - 0=0000 -> -0=1111
- Tatsächlich ist das sehr sinnvoll, denn es gilt -3 > -5. Dies gilt hier ebenfalls im Binären: 1100 > 1010

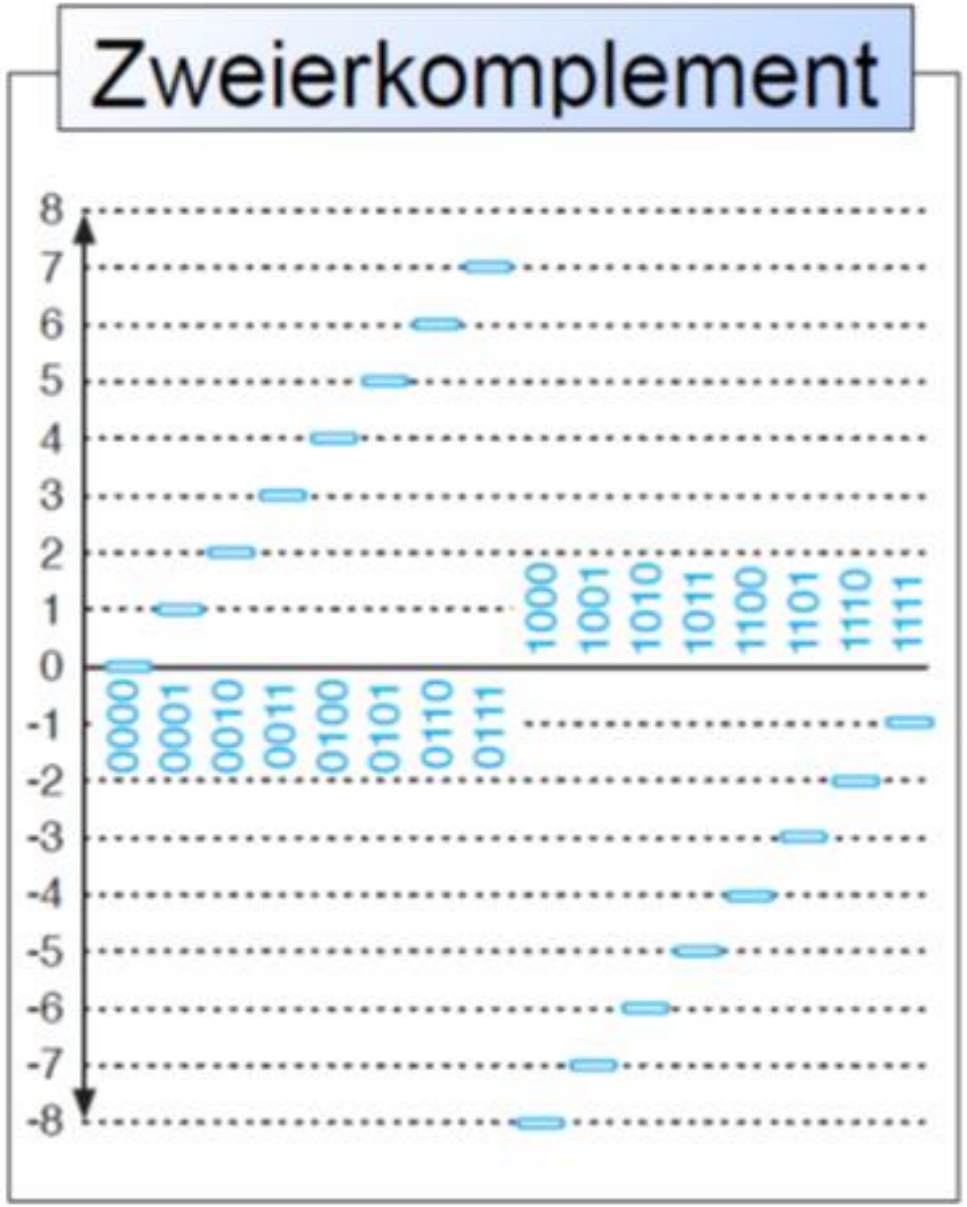


Zweierkomplement

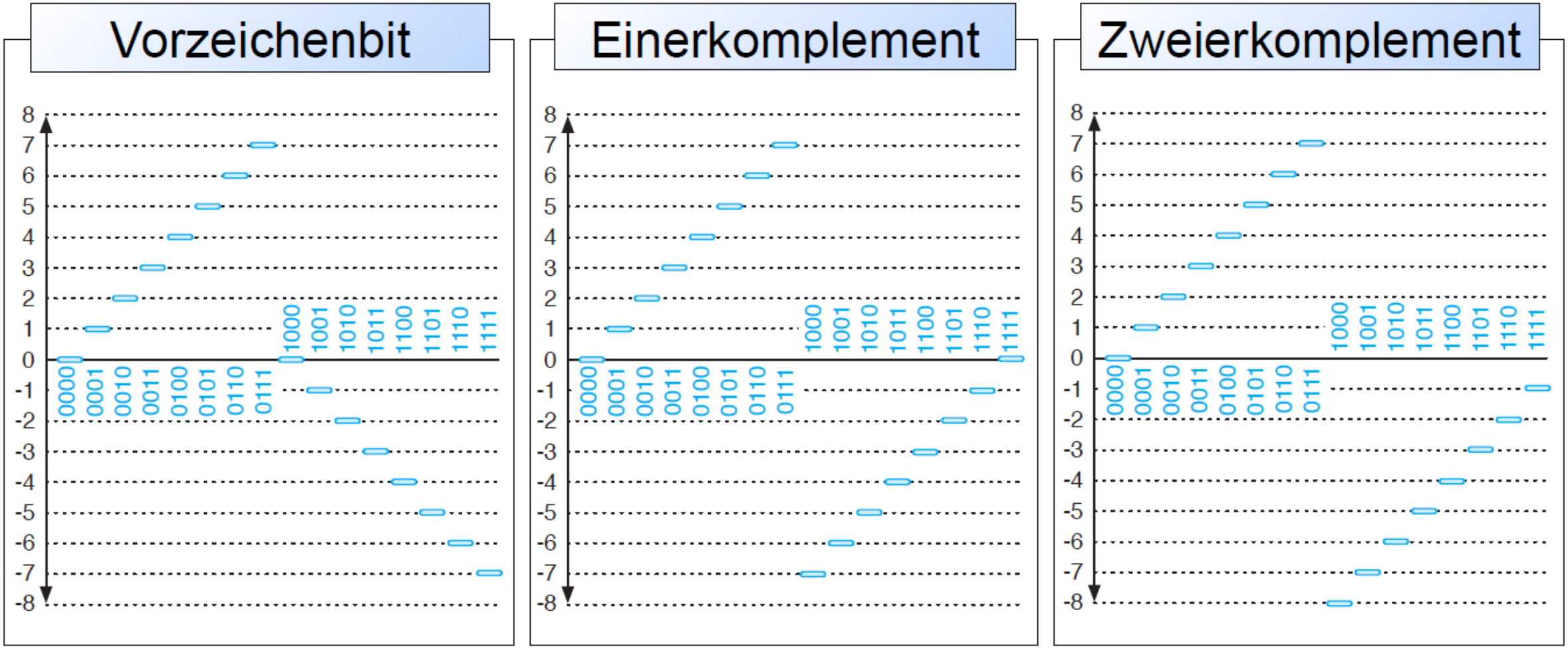
- Das erste Bit gibt auch hier das Vorzeichen an, aber negative Zahlen werden durch das Invertieren aller Bits + 1 gebildet.
- Beispiele:
  - -1: 1=0001 -> 1110 + 1 = 1111
  - -4: 4=0100 -> 1011 + 1 = 1100
  - -0 = 0, denn 0000 -> 1111 + 1 = 0000
  - Trick zum leichten Umrechnen: Die Negativ-1 kann hier als -8 gezählt werden.

    -8 +5 = -3

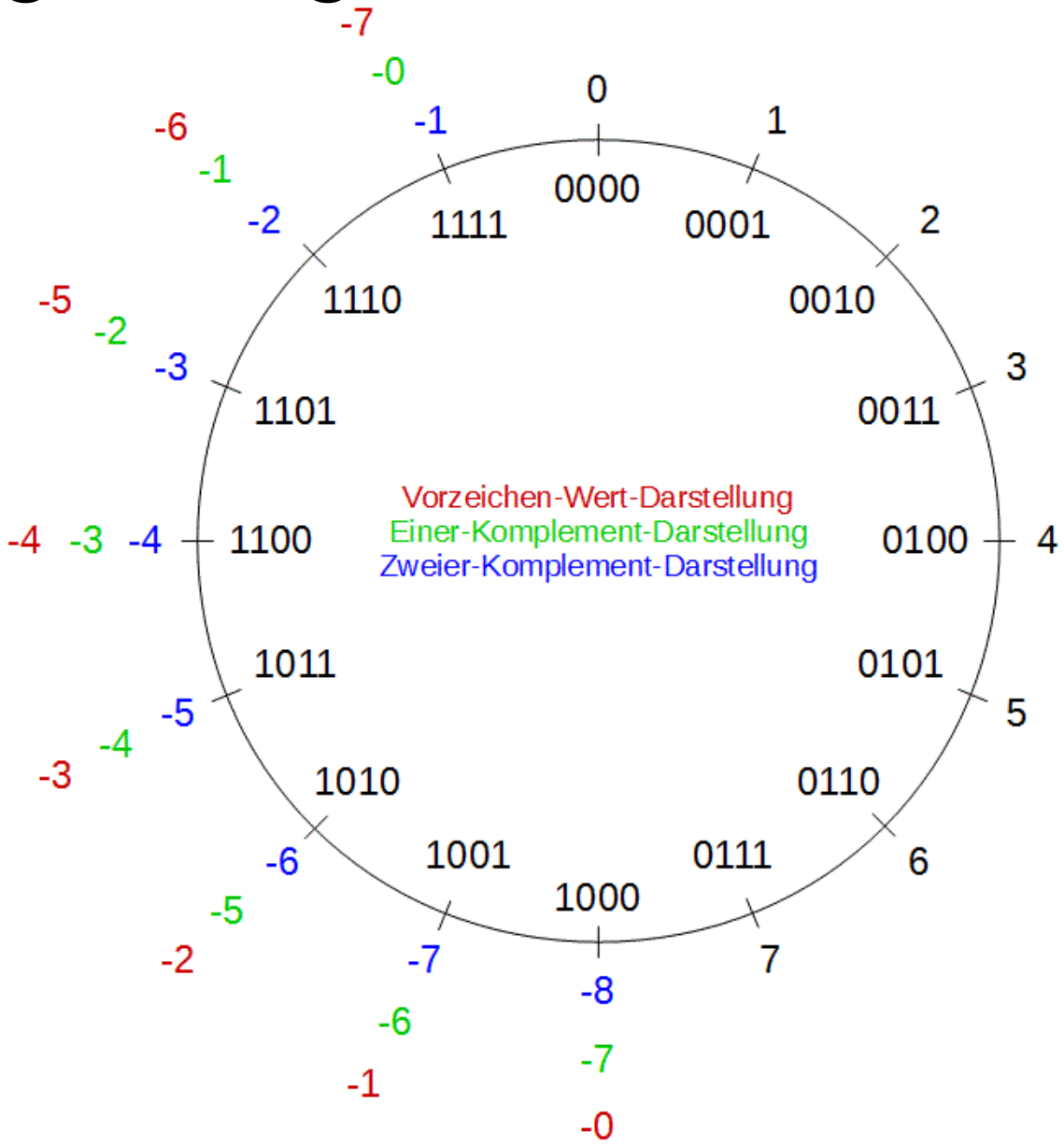
  - 1 1 0 1



Vorzeichenbit, Einerkomplement, Zweierkomplement



Kreisdarstellung der negativen Zahlen



Vorteil der Komplemente gegenüber dem Vorzeichenbit

- -2 + (+5) = +3
- Vorzeichenbitdarstellung

$$\begin{array}{r} 1010 \\ +0101 \\ =0011 \end{array}$$

Komplementdarstellung

$$\begin{array}{r} 1110 \\ +0101 \\ =0011 \end{array}$$

- Fazit: Die Komplementdarstellungen haben den großen Vorteil, dass man weiterhin normal im Stellenwertsystem rechnen kann, während bei der Vorzeichenbitdarstellung ständig geprüft werden müsste, ob ein Vorzeichenwechsel vorliegt, weil sonst die Rechnung nicht mehr stimmt.

Zusammenfassung

- Die Komplementdarstellungen haben den entscheidenden Vorteil, dass man weiterhin normal rechnen kann. Im Rechenwerk wird das Addieren wie das schriftliche Addieren auf Papier umgesetzt.
- Offensichtlich ist das Zweierkomplement das bessere System, da z.B. beim binären Hochzählen beim Einerkomplement darauf geachtet werden, dass die 0 nicht doppelt gezählt wird, während beim Zweierkomplement keine Probleme auftreten.
- Einerkomplement: 0=0000 und 1111, da 1111 der -0 entspricht
- Zweierkomplement: 0=0000, da -0: 0=0000 -> 1111 + 1 = 0000
- => Das Zweierkomplement ist die beste Variante!

Zweierkomplement in Java

- Das Zweierkomplement ist die übliche Darstellungsweise für positive und negative ganze Zahlen.
- Auch Java speichert Zahlen im Zweierkomplement ab:

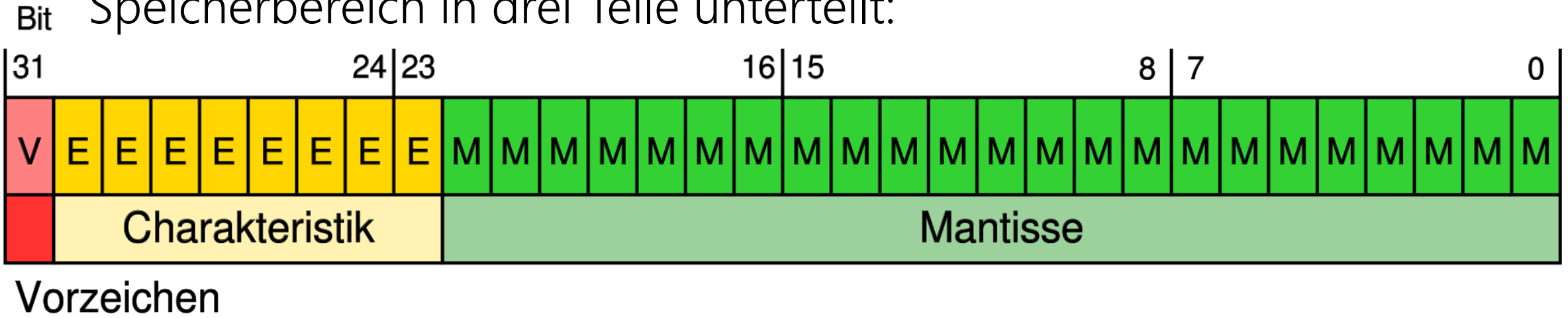
Typname	Größe[1]	Wrapper-Klasse	Wertebereich	Beschreibung
boolean	undefiniert[2]	java.lang.Boolean	true / false	Boolescher Wahrheitswert, Boolescher Typ[3]
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')	Unicode-Zeichen (UTF-16)
byte	8 bit	java.lang.Byte	-128 ... 127	Zweierkomplement-Wert
short	16 bit	java.lang.Short	-32.768 ... 32.767	Zweierkomplement-Wert
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647	Zweierkomplement-Wert
long	64 bit	java.lang.Long	-2^{63} bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$[4]	Zweierkomplement-Wert

Hexadezimalsystem

- Wir haben das Hexadezimalsystem bereits als ein Stellenwertsystem kennengelernt und damit auch Zahlen dargestellt und zurückgewandelt.
- Das Hexadezimalsystem wird sehr gerne genutzt, da es eine effiziente und auch besser lesbare Variante darstellt, um Binärzahlen zu lesen.
- Zusammenfassung von 4 Bit zu einer Stelle
  -> zweistellige Hexadezimalzahl entspricht einem Byte
  $(A3)_{16} = \underbrace{1010}_{A}\ \underbrace{0011}_{3} = (163)_{10}$
- Beispiel: Speicheradresse im Arbeitsspeicher: 0x106e1bcd

Dezimalzahlen

- Dezimalzahlen werden in der Regel nach dem Standard IEEE 754 dargestellt.
- Für die Darstellung einer solchen Gleitkommazahl mit 32 Bit für der Speicherbereich in drei Teile unterteilt:



- Das erste Bit gibt das Vorzeichen an. Die nächsten 8 Bit den Exponenten. Der Rest ist die Mantisse (=Als Mantisse bezeichnet man die Ziffernstellen einer Gleitkommazahl vor der Potenz.).

Darstellungsform einer 32-Bit-Gleitkommazahl

- Eine 32-Bit-Gleitkommazahl wird folgendermaßen dargestellt:
- $(-1)^{VZ} \cdot 1,M \cdot 2^{E-127}$

  - VZ: Vorzeichenbit (0 positiv; 1 negativ)
  - M: Mantisse
  - E: Exponent

Beispiel: Dezimalzahl → IEEE754-Gleitkommazahl

- 22,375
- $22_{10} \rightarrow 10110_2$ Umwandlung wie bereits bekannt

- Umwandlung der Nachkommastellen (Die Stelle vor dem Komma wird immer als Binärstelle genutzt):
- $0{,}375 \cdot 2 = 0{,}75$ -> 0 (Hochwertigste Stelle)
- $0{,}75 \cdot 2 = 1{,}5$ -> 1 (Stellenwertigkeit jetzt andersherum)
- $0{,}5 \cdot 2 = 1$ -> 1 (Niederwertigste Stelle)

=> $22{,}375_{10} = 10110{,}011_2$

Beispiel: Dezimalzahl → IEEE754-Gleitkommazahl

- $=> 22{,}375_{10} = 10110{,}011_2 = 1{,}0110011 \cdot 2^4$
- $1{,}0110011 \cdot 2^4 = 1{,}0110011 \cdot 2^{131-127}$ (IEEE-754-Form)

- VZ: 0, weil die Zahl positiv ist
- E: $131_{10} = 10000011_2$
- M: 0110011 Die 1 vor dem Komma wird nicht mitgespeichert (Hidden Bit; siehe Formel)
- 22,375 als Gleitkommazahl nach IEEE 754 single (32 Bit)
- $0\ 10000011\ 01100110000000000000000$

Aufgabe

- Gib die gegebenen Dezimalzahlen als Binärzahl nach der IEEE-754-Norm mit 32 Bit an:
- $-194{,}6875_{10}$
- $0{,}203125_{10}$

Beispiel: IEEE754-Gleitkommazahl → Dezimalzahl

- $0\ 10000011\ 01100110000000000000000$
- E: $10000011_2 = 131_{10} \rightarrow 131 - 127 = 4$ -> Um 4 Bit verschoben

- M: $0110011 \rightarrow \quad 1{,}0110011 \cdot 2^4 = 10110{,}011$ (Hidden Bit nicht vergessen!)
- Formel zur Umwandlung kann um die Nachkommastellen erweitert werden:
- $\ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + a_{-3} \cdot 2^{-3} \ldots$
- $=> 10110{,}011_2 = 22{,}375_{10}$

Aufgabe

- Gib die gegebenen Binärzahlen in der IEEE-754-Norm mit 32 Bit als Dezimalzahlen an:
- 1 01111011 01100000000000000000000
- 0 10001001 10100100111100000000000

Ungenauigkeiten bei der Umwandlung

- 0,4 ist im Binären eine periodische Zahl, da sich 0,4 = 2/5 sich nicht als Brüche der Art ½, ¼, 1/8 exakt darstellen lässt.
- $0{,}4_{10} = 0{,}\overline{0110}_2$
- Da der Speicherplatz der Mantisse begrenzt ist, muss nach diesen 23 Bits abgebrochen werden. Das heißt, dass 0,4 nicht exakt abgespeichert wird, sondern:
- 0,40000000596046447753906250->Umwandlungsfehler von 5.96E-9
- Das muss man im Hinterkopf behalten, bei Algorithmen, die sich einem Dezimalwert annähern, dass dieser möglicherweise nie exakt erreicht werden kann! -> Abbruchsbedingung, falls man eine Genauigkeit von z.B. 1E-6 hat.

IEEE-754 in Java

- Java speichert Dezimalzahlen im IEEE-754-Standard ab:

float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38	32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen.
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308	64-bit IEEE 754, doppelte Genauigkeit

- Beim Datentyp double (doppelte Größe von single) werden 64 Bit verwendet. Dadurch hat man folgende Aufteilung:
- 1 Bit Vorzeichen
- 11 Bit Exponent
- 52 Bit Mantisse
- Berechnung: $(-1)^{VZ} \cdot 1,M \cdot 2^{E-\mathbf{1023}}$