

1. Codierung

#DigitalTechnik

Themen

1. [Codierung Allgemein](#)
 2. [Arten von Codierung](#)
 3. [Zeichencodierung](#)
 4. [Zahlencodierung](#)
 5. [Signalcodierung](#)
-

Codierung Allgemein

Definition

Codierung ist die Darstellung von Information als Symbol oder Symbolfolge aus einem Alphabet.

Alphabet: eine endliche Menge von Symbolen

Hinweis

Die Ausgangsinfo kann analog oder digital sein, die Zielinfo ist immer digital.

Wenn die Ausgangsinfo digital ist, spricht man von einer "Umcodierung".

Arten von Codierung

1. **Zahlencodierung** -> Zahlenwerte werden dargestellt
2. **Zeichencodierung** -> Schriftzeichen werden dargestellt
3. **Signalcodierung** -> abstrakte Info wird als Signal oder Signalfolge dargestellt
4. **Anwendungscodierung** -> Info einer bestimmten Anwendung werden codiert
5. **Komprimierung** -> Umcodierung, bei welcher das Datenvolumen verringert werden soll

6. Verschlüsselung -> Umcodierung, bei welcher ohne Spezialinfos ("Schlüssel") die Ursprungsinformation nicht rekonstruiert werden kann

Definition zu 3.

Ein *Signal* ist eine physisch messbare Größe.

Beispiele zu 4.

- Textcodierung (HTML, DOCX, ODT,...)
- Bildcodierung (PNG, JPEG, GIF, SVG,...)
- Videocodierung (MP4,...)
- Audiocodierung (MP3,...)
- Farbcodierung (RGB, CMYK, HSL,...)

Zeichencodierung

Verschiedene Zeichencodierungen

- **ASCII** (≈ 1970): Alphabet = $\{0, 1, 2, 3, \dots, 127\}$
 - max. 128 Schriftzeichen darstellbar
- **ISO-8859** (≈ 1990): Alphabet = $\{0, 1, 2, 3, \dots, 255\}$
 - um "nationale Sonderzeichen" erweiterter ASCII-Zeichensatz
 - z.B. ä, ö, ü, Ä, Ö, Ü, ß, à, Á ç, ...
 - max. 256 Schriftzeichen darstellbar
 - verschiedene Varianten:
 - ISO-8859-1: westeuropäisch (veraltet)
 - ISO-8859-5: kyrillisch
 - ISO-8859-7: griechisch
 - ISO-8859-15: westeuropäisch (inkl. €)
- **Unicode** (≈ 2000): Anspruch, alle Schriftzeichen, aller aktuellen, ehemaligen und zukünftigen Schriftsprachen darstellen zu können
 - notwendig ist eine bestimmte "*Transfercodierung*":

#überschlägigesRechnen
 - **UTF-8**: Alphabet = $\{0, \dots, 255\}$
 - **UTF-16**: Alphabet = $\{0, \dots, 65535\}$
 - **UTF-32**: Alphabet = $\{0, \dots, 2^{32} - 1\}$

- Es muss jeweils mindestens ein "Fortsetzungssymbol" geben, welches anzeigt dass die Symbolfolge für das aktuelle Schriftzeichen noch nicht zu Ende ist
- Die Wahl zwischen UTF-8/-16/-32 ist eine Frage der verwendeten Schriftzeichen und der Speicherplatz effizienz

Zahlencodierung

Abzählssysteme

Definition

Abzählssysteme grundsätzlich: dargestellter Zahlenwert wird bestimmt als Summe der dargestellten Symbolwerte

Fingerabzählssystem

Alphabet = { **Finger** 🖐 }

Symbolwert (**Finger** 🖐) = 1

Beispiele:

- 5 -> 🖐
- 2 -> 🖐

Vor- und Nachteile

⊕ extrem einfach, verständlich

⊖ extrem eingeschränkt, Wertebeschränkt (0-10)

(bis auf weiteres nur nicht negative und ganze Zahlen)

⊕ extrem einfache Verfahren für Addition und Subtraktion

einfache Strichliste

Alphabet = { | }

Symbolwert (|) = 1

? Vor- und Nachteile

⊕ unendlicher Wertebereich

⊖⊕ Schreibwerkzeug wird benötigt

⊖ übersichtlicher Wertebereich ist eingeschränkt

⊕⊕⊖ Addition extrem einfach aber Subtraktion erfordert "Löschmöglichkeit"

⊖⊖⊕ Multiplikation und Division mit einfachen Verfahren (mehrfache Addition bzw. Subtraktion) möglich, aber deutlich höherer Aufwand

Beispiele:

5: |||||

2: ||

7: ||||| (oder auch |||| |)

10: ||||| (oder auch |||| ||)

erweiterte Strichliste

Jeder fünfte Strich wird als Querstrich durch vier Striche gezogen

Alphabet = { |, + + + + }

Symbolwert (|) = 1

Symbolwert (+ + + +) = 5

Regel: Symbole müssen sortiert nach Wertigkeit notiert werden.

? Vor- und Nachteile

⊖ übersichtlicher Wertebereich bis 50

⊖ Addition und Subtraktion erfordern zusätzliche Neusortierung und ggf. Zusammenfassen bzw. Auflösung von Symbolen

römisches Zahlensystem

Alphabet = { I, V, X, L, C, D, M }

Symbolwert (I) = 1

Symbolwert (V) = 5

Symbolwert (X) = 10

Symbolwert (L) = 50

Symbolwert (C) = 100

Symbolwert (**D**) = 500

Symbolwert (**M**) = 1000

Beispiele:

4 \neq IIII

4 = IV

Sonderregel: niedrigwertiges Symbol vor höherwertigem Symbol ist manchmal erlaubt, aber der niedere Wert wird dann vom höheren Wert abgezogen und nicht aufsummiert. Nur maximal drei gleiche Symbole nebeneinander erlaubt (daraus folgt auch einer der Nachteile des Systems).

❓ Vor- und Nachteile

⊖ nur endlicher Wertebereich bis etwa 4000

⊖ Rechnen ist ein Alptraum

Stellenwertsysteme (SWS)

Dezimalsystem

Alphabet = {**0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**}

Symbole heißen "Ziffern"

Ziffernwert (**0**) = 0

.

Ziffernwert (**9**) = 9

n -stellige Zahl ist eine Folge von Ziffern

$Z_{n-1}Z_{n-2}Z_{n-3} \dots Z_2Z_1Z_0$

Werteformel

Wert ($Z_{n-1} \dots Z_0$) =

$$\sum_{i=0}^{n-1} |Z_i \cdot 10^i$$

Z_i = Ziffernwert

10^i = Stellenwert

❓ Vor- und Nachteile

⊖⊕ etwas komplexer, aber noch etwas Einarbeitungszeit gut verständlich und einfach verwendbar

⊕ unendlicher Wertebereich

⊖⊕ sehr großer übersichtlicher Wertebereich (bis $10^{10} = 10\text{Mrd.}$)

⊕ erstmals explizite Darstellung der "0" möglich

⊖⊕ Verfahren für alle Grundrechenarten mit gewisser Komplexität, aber mäßigem Aufwand verfügbar

SWS zur Basis b

$b \in \mathbb{N} \setminus \{1\} \quad (b > 1 \text{ oder } b \geq 2)$

Ziffernmenge enthält genau b verschiedene Ziffern, die kleinste Ziffer hat den Wert 0, die größte Ziffer hat den Wert $(b - 1)$

Werteformel

Wert $(Z_{n-1}Z_{n-2} \dots Z_2Z_1Z_0) =$

$$\sum_{i=0}^{n-1} |Z_i| \cdot b^i$$

Hinweis

SWS zur Basis $b = 1$ ist **keine** Strichliste; der einzig darstellbare Wert ist 0, da 0 auch die einzige Ziffer ist -> SWS zur Basis 1 macht keinen Sinn

Vor- und Nachteile

⊕ übersichtlicher Wertebereich bis etwa b^{10} (exponentielle und nicht nur lineare Abhängigkeit von b)

Hinweis: Bei größeren Basen leidet die Übersichtlichkeit, aber an der Anzahl

gängige Basen

$b = 10$: Dezimalsystem -> von Menschen verwendet

$b = 2$: Binär/Dualsystem -> von Computern verwendet

$b = 16$: Hexadezimalsystem -> für kompakte und Computer-nahe Darstellung von

Zahlen mit einfacher und direkter
Umrechnungsmöglichkeit ins Binärsystem
 $b = 8$: Oktalsystem -> folgt in Kürze

Umrechnung zwischen verschiedenen Basen

a) Umrechnung von Basis $b \neq 10$ nach Basis 10

-> Werteformel

$$\sum_{i=0}^{n-1} |Z_i| \cdot b^i$$

b) Umrechnung von Basis nach Basis $b_2 \neq 10$

1. umgekehrte Werteformel

2. Ganzzahldivision und Restbildung

die notierten Reste ergeben die Ziffernfolge zur Basis b_2

Beispiel:

$$62_{10} = ?_2$$

$$62 : 2 = 31 \text{ R } 0 \quad 0 = Z_0$$

$$31 : 2 = 15 \text{ R } 1 \quad 1 = Z_1$$

$$15 : 2 = 7 \text{ R } 1 \quad 1 = Z_2$$

$$7 : 2 = 3 \text{ R } 1 \quad 1 = Z_3$$

$$3 : 2 = 1 \text{ R } 1 \quad 1 = Z_4$$

$$1 : 2 = 0 \text{ R } 1 \quad 1 = Z_5$$

$$= 111110_2$$

$$\begin{aligned} 111110_2 &= 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 \\ &= 0 \cdot 1 + 1 \cdot 2 + 1 \cdot 4 + 1 \cdot 8 + 1 \cdot 16 + 1 \cdot 32 \\ &= 0 + 2 + 4 + 8 + 16 + 32 \\ &= 63 \end{aligned}$$

$$62_{10} = ?_3$$

$$62 : 3 = 20 \text{ R } 2 \quad 2 = Z_0$$

$$20 : 3 = 6 \text{ R } 2 \quad 2 = Z_1$$

$$6 : 3 = 2 \text{ R } 0 \quad 0 = Z_2$$

$$2 : 3 = 0 \text{ R } 2 \quad 1 = Z_3$$

$$3 : 2 = 1 \text{ R } 1 \quad 1 = Z_4$$

$$1 : 2 = 0 \text{ R } 1 \quad 1 = Z_5$$

$$= 2022_3$$

$$\begin{aligned} 2022_3 &= 2 \cdot 3^0 + 2 \cdot 3^1 + 0 \cdot 3^2 + 2 \cdot 3^3 \\ &= 2 \cdot 1 + 2 \cdot 3 + 0 \cdot 9 + 2 \cdot 27 \\ &= 2 + 6 + 0 + 54 \\ &= 62_{10} \end{aligned}$$

c) Umrechnung von Basis $b_1 \neq 10$

allgemein: Umrechnung in zwei Teilschritten:

von b_1 nach $b_z = 10$

und von $b_z = 10$ nach b_2

(reintheoretisch wäre das auch per Ganzzahldivision durch b_2 direkt machbar, aber die Rechnung müsste zur Basis b_1 durchgeführt werden)

direkte Umrechnung ist möglich, falls $b_1^k = b_2^m$, dann könnte k Ziffern zur Basis b_1 in eine Ziffer zur Basis b_2 "umgerechnet" werden (am besten in einer Tabelle)

Beispiel:

Von $b_1 = 2$ nach $b_2 = 16$ (wird umgerechnet):

$$2^4 = 16$$

direkte Umrechnung ist auch dann möglich wenn $b_1^k = b_2^m$, dann kann eine Folge von k Ziffern zur Basis b_1 direkt in eine Folge von m Ziffern zur Basis b_2 umgerechnet werden

Beispiel:

$$8^k = 16^m$$

$$(2^3)^k = (2^4)^m$$

$$2^{3k} = 2^{4m}$$

$$3k = 4m$$

-> $k = 4$ und $m = 3$

$$8^4 = 16^3$$

Problem: Umrechnungstabelle hat $8^4 = 16^3 = 2^{12} = 4096$

Beispiel:

Wert	Hexziffern	4-Binärziffernfolge
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101

Wert	Hexziffern	4-Binärziffernfolge
14	E	1110
15	F	1111

$$AFFE_{16} = 1010\ 1111\ 1111\ 1110_2$$

$$10\ 1111\ 1100_2 = 2FC_{16}$$

Hexadezimal	Oktal
⊕ kompakte Darstellung	⊕ nur Verwendung von üblichen Ziffersymbolen
⊕ 1 Byte ist mit genau zwei Hex-Ziffern darstellbar	
heute üblicherweise verwendet	früher häufiger verwendet

Wert	Oktalziffern	3-Binärziffernfolge
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

$$AFFE_{16} = 1010\ 1111\ 1111\ 1110_2 = 001\ 010\ 111\ 111\ 111\ 110_2 = 127776_8$$

Aufhebung der Einschränkungen:

Darstellung von (auch) negativen Zahlen

a) Vorzeichen und Betrag

notwendig ist (mindestens) ein weiteres Symbol im Alphabet für das "Vorzeichen":

Minus: -

optional Plus: +

⊖ Nachteile:

- zwei unterschiedliche Verfahren für die "Addition" von positiven und negativen Zahlen notwendig (Addition einer negativen Zahl müsste als Subtraktion ausgeführt werden)
- nicht eindeutige Darstellung der Null (+0) und (-0)

b) Einerkomplement (nur noch Binär)

Festlegen der Stellenzahl so, dass sie mindestens "1" größer ist als notwendig für den größten Betrag im gesamten Rechenweg

Invertieren jeder Ziffer in der Ziffernfolge:

$0 \rightarrow 1$ und $1 \rightarrow 0$

Beispiel:

$$5_{10} = 0101_2 \xrightarrow{inv} 1010 = -5$$

$$42_{10} = 101010_2 = 0010\ 1010 \xrightarrow{inv} 1101\ 0101 = -42$$

$$13_{10} = 1101_2 = 0000\ 1101 \xrightarrow{inv} 1111\ 0010 = -13$$

$$\begin{array}{r} -42 : 1101\ 0101 \\ +13 : 0000\ 1101 \\ \hline 1110\ 0010 = -29 \text{ negativ} \end{array}$$

$$1110\ 0010 \xrightarrow{inv} 0001\ 1101 = 29$$

$$\begin{array}{r} +42 : 0010\ 1010 \\ -13 : 1111\ 0010 \\ \hline 0001\ 1100 = 28 \text{ positiv} \end{array}$$

Ergebnis liegt (haarscharf) um 1 daneben, ist also falsch. Die Hälfte aller Rechnungen im Einerkomplement ist richtig, die andere falsch.

negierte Null

$$0 = 0000\ 0000 \xrightarrow{inv} 1111\ 1111 = "-0"$$

→ keine eindeutige Darstellung der Null

Wertebereich mit 8bit

$$\text{größte Zahl: } 0111\ 1111 = 1 + 2 + 4 + 8 + 16 + 32 + 64 = 127$$

$$\text{kleinste Zahl: } 1000\ 0000 \xrightarrow{inv} 0111\ 1111 = 127 \implies 1000\ 0000 = -127$$

- Wertebereich von -127 bis 127 mit 8bit Einerkomplement darstellbar.
- insgesamt "nur" 255 Zahlendarstellbar (statt 256, wie mit 8bit eigentlich möglich sein sollten)

→ somit ergeben sich 3 Nachteile des 1er-Komplements

c) Zweierkomplement

Bildung wie 1er-Komplement, d.h. Stellenanzahl festlegen, alle Ziffern "invertieren", dann zusätzlich "+1" addieren.

$$42 : 101010 = 0010\ 1010 \xrightarrow{inv} 1101\ 0101 \xrightarrow{+1} 1101\ 0110$$

$$13 : 1101 = 0000\ 1101 \xrightarrow{inv} 1111\ 0010 \xrightarrow{+1} 1111\ 0011$$

$$\begin{array}{r} - 42 : 1101\ 0110 \\ + 13 : 0000\ 1101 \\ \hline 1110\ 0011 \text{ negativ} = -29 \end{array}$$

$$1110\ 0011 \xrightarrow{inv} 0001\ 1100 \xrightarrow{+1} 0001\ 1101 = 1 + 4 + 8 + 16 = 29$$

$$+ 42 : 0010\ 1010$$

$$- 13 : 1111\ 0011$$

$$\begin{array}{r} \text{ign.} \rightarrow 1\ 1100\ 010 \\ \hline 0001\ 1101 = 29 \text{ positiv} \end{array}$$

⚠ Wichtig

Die führende 0 gibt an, dass es sich um eine positive Zahl handelt. Ist die führende Ziffer eine 1, so ist die Zahl negativ.

💧 Darstellung der 0

$$0000\ 0000 \xrightarrow{inv} 1111\ 1111 \xrightarrow{+1} 0000\ 0000 = 0$$

- eine Darstellung der 0

💧 Wertebereich mit 8bit

größte Zahl: $0111\ 1111 = 127$

kleinste Zahl: $1000\ 0000 \xrightarrow{inv} 0111\ 1111 \xrightarrow{+1} 1000\ 0000$ (nicht negativ, da Betrag einer Zahl!) $= 128 \implies -128$

- Wertebereich geht von -128 bis +127
- insgesamt 256 verschiedene Werte
- ⊖ unsymmetrischer Wertebereich, d.h. es gibt mehr negative als positive darstellbare Zahlen

Darstellung (auch) nicht-ganzer Zahlen

(vorläufig wieder nur nicht-negative Zahlen)

a) **Brüche** (d.h. Darstellung mit Zähler und Nenner, dazwischen ein Bruchstrich)

z.B. $\frac{1}{2}, \frac{1}{4}, \frac{1}{10}, \frac{3}{8}$

$$1 = \frac{1}{1} = \frac{2}{2} = \frac{3}{3} = \frac{4}{4} = \dots$$

→ für jede Zahl gibt es unendlich viele Darstellungen als Bruch

Abhilfe: maximales Kürzen

Darstellung als Bruch ist im Computer unüblich.

Ausnahme: Mathematikprogramme zum Lösen von Gleichungssystemen (z.B. Maple, GeoGebra, WolframAlpha)

b) **Festkommazahl**

notwendig ist (mindestens) ein weiteres Symbol im Alphabet:

Komma: ,

→ Das Komma darf nur genau einmal verwendet werden, davor gibt es n Vor- und m

Nachkommastellen $Z_{n-1}Z_{n-2} \dots Z_2Z_1Z_0, Z_{-1}Z_{-2} \dots Z_{-m+1}Z_{-m}$

Das geht mit unterschiedlichen Basen $b \in \mathbb{N} \setminus \{1\}$

Kommazahlen zwischen verschiedenen Basen umrechnen:

z.B. $1101,0011_2 = ?_{10}$

Werteformel:
$$\sum_{i=0}^{n-1} |Z_i| \cdot b^i + \sum_{j=1}^m |Z_j| \cdot b^{-j} = \sum_{i=-m}^{n-1} |Z_i| \cdot b^i$$

$$\begin{aligned} 1101,0011_2 &= 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= 1 + 4 + 8 + \frac{1}{8} + \frac{1}{16} \\ &= 13 + 0,125 + 0,0625 \\ &= 13,1875_{10} \end{aligned}$$

Umrechnung von Basis 10 nach Basis $b \neq 10$:

getrennte Behandlung von Vor- und Nachkommaanteil:

- Vorkommateil umwandeln, wie bekannt, also Ganzzahldivision mit Rest oder umgekehrte Werteformel.
- Nachkommateil entweder über umgekehrte Werteformel (teilweise schwierig handhabbar) oder über Multiplikation mit Zielbasis und Aufteilung in Vor- und Nachkommaanteil; Vorkommaanteil ist die nächste Nachkommastelle, mit dem Nachkommateil muss weitergerechnet werden.

$$13,1875_{10} = 1101,00110_2$$

$$0,1875 \cdot 2 = 0,375 \rightarrow Z_{-1} = 0$$

$$0,375 \cdot 2 = 0,75 \rightarrow Z_{-2} = 0$$

$$0,75 \cdot 2 = 1,5 \rightarrow Z_{-3} = 1$$

$$0,5 \cdot 2 = 1,0 \rightarrow Z_{-4} = 1$$

$$0,0 \cdot 2 = 0,0 \rightarrow Z_{-5} = 0$$

...

c) Gleitkommazahl (GKZ)

GKZ besteht aus Mantisse und Exponent.

$$\text{Wert(GKZ)} = \text{Mantisse} \cdot b^{\text{Exponent}}$$

Die Mantisse ist eine (Fest-)Kommazahl und gibt die Ziffernfolge der Zahl an.

Der Exponent ist eine ganze Zahl (positiv und negativ möglich) und gibt die Verschiebung des Kommas (Anzahl der Stellen) bei der Mantisse an (Vorzeichen des Exponenten steht dabei für die Richtung der Verschiebung nach rechts oder links).

$$\text{z.B. } 1 = 1,0 \cdot 10^0 = 0,1 \cdot 10^1 = 10,0 \cdot 10^{-1} = \dots$$

→ für jede Zahl gibt es unendlich viele Darstellungen als GKZ

Abhilfe: normierte Darstellung, aber: es gibt zwei Varianten

a) nur eine Vorkommastelle, diese ist $\neq 0$

b) Vorkommaanteil ist $= 0$, die erste Nachkommastelle ist $\neq 0$

Die gängige Variante ist Variante a

Problem: Für die "0" gibt es keine normierte Darstellung

Abhilfe: Für die "normierte 0" gibt es eine reservierte Bitfolge (nur "0")

Hinweis

Die Basen für die Darstellung der Mantisse und mit welcher der Exponent potenziert wird, müssen die gleichen sein, da der Exponent die Kommaverschiebung der Mantisse vorgeben soll.

Im Computer wird die Basis $b = 2$ verwendet.

Größenvergleich von zwei GKZ

1. Vergleich der Exponenten → größerer Exponent gehört zur größeren Zahl

2. bei gleichem Exponent → erst Größenvergleich der Mantissen

Beispiel zum Größenvergleich:

$$42 = 0010\ 1010 \xrightarrow{\text{inv}} 1101\ 0101 \xrightarrow{+1} 1101\ 0110 = -42$$

$$13 = 0000\ 1101 \xrightarrow{\text{inv}} 1111\ 0010 \xrightarrow{+1} 1111\ 0011 = -13$$

→ Prüfen von links nach rechts, erste unterschiedliche Ziffer: 1 gehört zur größeren Zahl

+42 = 0010 1010

-13 = 1111 0011

→ Problem: bei unterschiedlichem Vorzeichen gehört die "0" zur größeren Zahl

- Fallunterscheidung wäre nötig
- *Keine 2er-Komplement-Darstellung beim Exponent üblich!*

Stattdessen: "Bias-Darstellung"

$$\text{Exp}_{\text{gesp}} = \text{Exp}_{\text{real}} + \text{Bias}$$

$$\text{Exp}_{\text{real}} = \text{Exp}_{\text{gesp}} - \text{Bias}$$

z.B. bei 8bit → Bias = 127 → Wertebereich für Exp_{real} von -127 bis +128 darstellbar

Normierung (Variante a): eine Vorkommastelle $\neq 0$

im Binärsystem ist diese also immer "1"

→ diese "1" braucht man nicht explizit zu speichern, sondern nutzt das bit besser für eine weitere Nachkommastelle.

→ Hidden-Bit-Darstellung (üblicherweise verwendet und empfohlen)

Problem: Bei der "0" darf es kein Hidden-Bit geben → beim für "0" definierten Bitmuster gibt es kein Hidden-Bit

Darstellung (auch) nicht-ganzer, negativer Zahlen

eine negative Mantisse macht auch die Gleitkommazahl negativ.

Addition von 2 GKZ:

- erst muss bei einer GKZ das Komma so verschoben werden, dass sie denselben Exponenten wie die andere GKZ hat (Aufhebung der Normierungsbedingung für eine GKZ) danach Addition der Mantisse.
z.B.

$$1,01101 \cdot 2^3 = 0,0101101 \cdot 2^5$$
$$1,01101 \cdot 2^5$$

$$01,01101 \xrightarrow{inv} 10,10010 + 10,10011 \cdot 2^3$$

→ zur Fallunterscheidung müsste jetzt eine führende "1" vorne drangestellt werden

→ Fallunterscheidung wird eingespart

negative Mantisse?

- kein 2er-Komplement
- Vorzeichen und Betrag als Speicherformat (nur für die Rechnung wird "kurzzeitig" das 2er-Komplement verwendet)

IEEE 754

Genauigkeit	Speichermenge [bit]	Vorzeichen [bit]	Mantisse [bit]	Exponent [bit]	empfohlener Bias
half	16	1	10	5	15
float/single	32	1	23	8	127
double	64	1	52	11	1023

außerdem:

- Hidden-Bit-Darstellung empfohlen
- Normierung mit einer Vorkommastelle $\neq 0$ empfohlen
- gespeichert wird in der Reihenfolge: Vorzeichen - Exponent - Mantisse

reservierte Bitmuster

- alle bit "0": Zahlenwert 0 (kein Hidden-Bit)
- alle Exponentenbit "0": kein Hidden-Bit, keine normierte Darstellung
 $\text{Exp}_{\text{real}} = 1 - \text{Bias}$ (um Lücke zwischen kleinster normierter und größter nicht-normierter Zahl zu verhindern)
- alle Exponentenbita "1"
 - und alle Mantissenbits "0": Zahl liegt außerhalb des darstellbaren Wertebereichs also " $\pm\infty$ "
 - und Mantisse $\neq 0$: Zahl ist "NaN" (z.B. beim Teilen durch 0 oder Ziehen der Wurzel von -1)

Wertebereich von Gleitkommazahlen — am Beispiel von "half"

zum Vergleich:

- 16 bit nicht-negative ganze Zahlen: 0 bis 65535 ($= 2^{16} - 1$)
- 16 bit 2er-Komplement: -32768 bis + 32767
- 16 bit GKZ "half":
 - größte Zahl (siehe [Tabelle](#)):
 - Wert (größte Zahl): $(2 - 2^{10}) \cdot 2^{15} = 2^{16} - 2^5 = 65536 - 32 = 65504$
 - kleinste Zahl (siehe [Tabelle](#)):
 - Wert (kleinste Zahl): -65504

half (größte Zahl)

0	11110	[1,]1111111111
---	-------	----------------

0	11110	[1,]1111111111
Vorzeichen positiv	Exponent (gespeichert) Exp (gesp) = 30 Exponent (real) = Exp(gesp) - Bias = 30 - 15 = 15	Mantisse Mantisse = $2 - 2^{10} = 2 - \frac{1}{1024}$

half (kleinste Zahl)

1	11110	[1,]1111111111
Vorzeichen negativ	Exponent (gespeichert) Exp (gesp) = 30 Exponent (real) = Exp(gesp) - Bias = 30 - 15 = 15	Mantisse Mantisse = $2 - 2^{10} = 2 - \frac{1}{1024}$

Hidden-Bit

kleinste positive Zahl (d.h. kleinster Betrag)

a) normierte Darstellung

0	00001	[1,]0000000000
VZ	Exp	Mantisse
	Exp _{gesp} = 1 Exp _{real} = 1 - 15 = -14	Mantisse = 1

$$\text{Wert (Zahl)} = 1 \cdot 2^{-14} = \frac{1}{16384}$$

b) nicht-normierte Darstellung

0	00000	[0,]0000000001
VZ	Exp	Mantisse
	Exp _{gesp} = 1 Exp _{real} = 1 - 15 = -14	Mantisse = 2^{-10}

$$\text{Wert (Zahl)} = 2^{-10} \cdot 2^{-14} = 2^{-24} \approx \frac{1}{16 \text{ Mio}}$$

kein Hidden-Bit

Umrechnung einer Dezimalkommazahl in eine Gleitkommazahl zur Basis 2

z.B. $-4,2 \cdot 10^{-1}$

1. Entfernen des Vorzeichens und merken für später: $4,2 \cdot 10^{-1}$
2. Umwandeln in eine Kommazahl ohne Exponent: 0,42
3. Umwandeln der Zahl (zur Basis 10) in die Zielbasis nach bekannten Verfahren, bis die Anzahl relevanter Stellen bei der Mantisse (d.h. die entsprechende Stellenzahl nach der ersten "1") erreicht ist

$$0,42_{10} = 0,011010111000_2 = 1,10101110000 \cdot 2^{-2}$$

$$0,42 \cdot 2 = 0,84$$

$$0,84 \cdot 2 = 1,68$$

$$0,68 \cdot 2 = 1,36$$

$$0,36 \cdot 2 = 0,72$$

$$0,72 \cdot 2 = 1,44$$

$$0,44 \cdot 2 = 0,88$$

$$0,88 \cdot 2 = 1,76$$

$$0,76 \cdot 2 = 1,52$$

$$0,52 \cdot 2 = 1,04$$

$$0,04 \cdot 2 = 0,08$$

⋮

4. Bestimmen des Exponenten durch verschieben des Kommas bei der Mantisse, um den Exponenten Exp_{real} zu bestimmen

$$0,011010111000_2 = 1,10101110000 \cdot 2^{-2}$$

$$\text{Exp}_{\text{real}} = -2$$

5. $\text{Exp}_{\text{gesp}} = \text{Exp}_{\text{real}} + \text{Bias}$

$$\text{Exp}_{\text{gesp}} = -2 + 15 = 13$$

6. Exp_{gesp} in Binärsystem umrechnen und mit vorgegebener Stellenzahl darstellen

$$\text{Exp}_{\text{gesp}} = -2 + 15 = 13 = 8 + 4 + 1 = 01101$$

7. Bitmuster in vorgegebener Reihenfolge notieren Vorzeichen – Exp_{gesp} – Mantisse

$$-4,2 \cdot 10^{-1} = 1 \ 01101 \ 1010111000$$

Vorzeichen, Exponent, Mantisse

Zahlencodierung lässt sich unterscheiden in

- Wertecodierung:
Der Wert an sich wird umgerechnet in die Zieldarstellung
- Zifferncodierung:
Die bereits in einem Stellenwertsystem codierte Zahl wird Ziffer für Ziffer in die andere Darstellung gebracht

Alle Zahlencodierungen waren bisher Wertecodierung

Ausnahme: direkte Umrechnung zwischen zwei verschiedenen Basen, falls $b_1^n = b_2^n$

weiteres Beispiel für Zifferncodierung: Darstellung von Zahlen im Fließtext

Nachteil: viel größerer Speicherumfang; oder mit bestimmter Anzahl bit lässt sich ein viel kleinerer Wertebereich darstellen als bei binär codierten Zahlen

Bsp.:

8bit in Textcodierung: Zahlen von 0 bis 9

8bit Binärcodierung: Zahlen von 0 bis 255

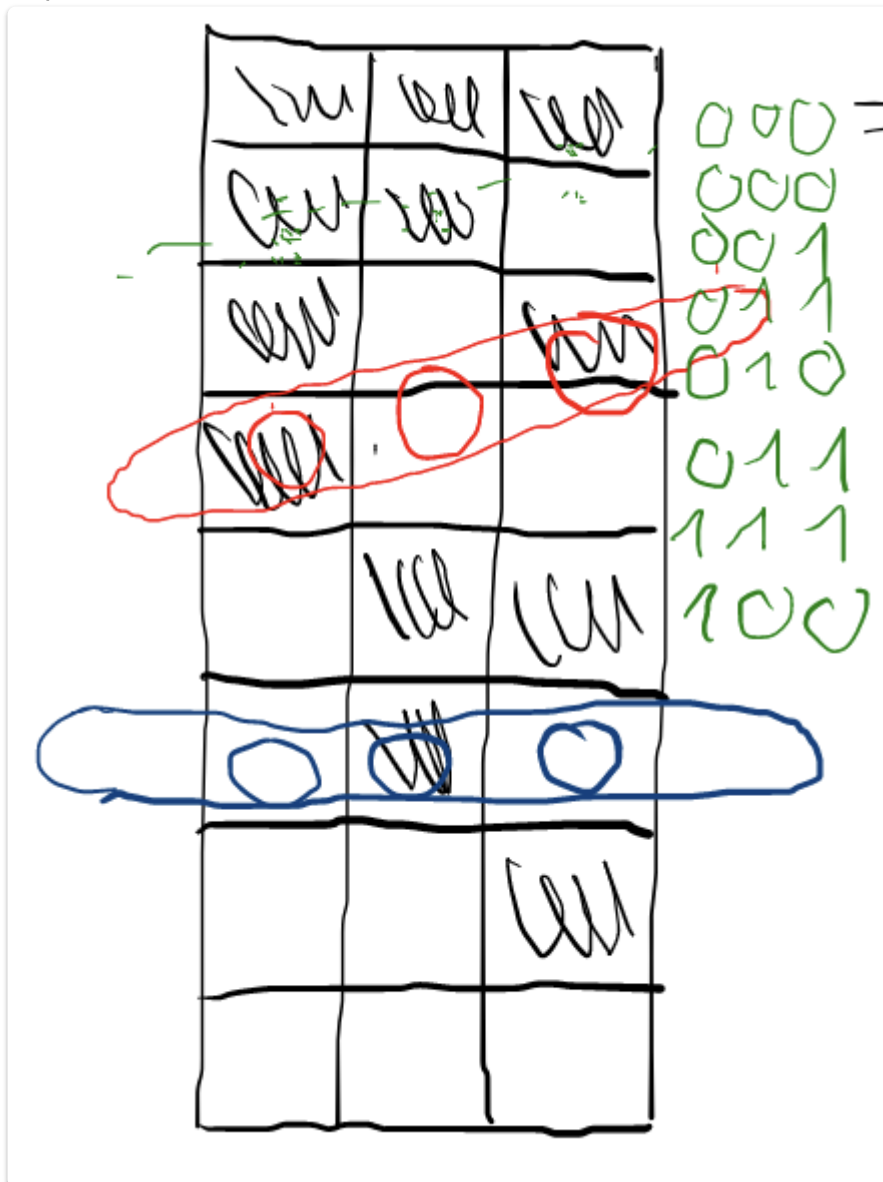
Besser: BCD (Binary Coded Decimals)

→ 4bit je Dezimalziffer reichen aus

Bsp: Bei ABAP (SAP-Prog.-Sprache) gibt es einen entsprechenden Datentyp sowie zugehörige Rechenroutinen

Nr.	4bit	Dez. Ziff
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	X
11	1011	X
12	1100	X
13	1101	X
14	1110	X
15	1111	X

X : nicht standardisiert, also entweder "Fehler" oder selbst anderen "wichtigen" Zeichen zugeordnet, z.B. Vorzeichen, Komma, ...



Falsche Zwischenwerte können immer dann entstehen, wenn zwischen zwei aufeinanderfolgenden Zahlenwerten sich mehr als eine Ziffer (quasi gleichzeitig) ändern müsste, die Änderung aber nacheinander vollzogen wird.

Definition

Ein einschrittiger Zahlencode ist eine Form der Zahlencodierung, bei welcher sich zwischen zwei aufeinanderfolgenden Zahlenwerten nur ein Symbol (eine Stelle, eine Ziffer) ändert.

Bsp.: Der Gray-Code ist ein einschrittiger Code

Gray-Code	Dezimalsystem	Gray-Code	Dezimalsystem
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11
0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Bildungsregel:

- Variante **a**: Ausgehend von "00...0" für 0 wird die Symbolfolge für den nächsten Wert gebildet, in dem das rechteste mögliche Symbol verändert wird, bei welchem eine noch nicht verwendete Symbolfolge entsteht.
- Variante **b**: aus einem n -stelligen kann man einen $n + 1$ -stelligen Gray-Code machen, indem man vor die bisherige Symbolfolge eine 0 stellt und für die neuen Symbolfolgen die alten Symbolfolgen in umgekehrter Reihenfolge mit vorangestellter "1" verwendet.

Signalcodierung

Definition

Darstellen von "abstrakten Infos" (typischerweise Bitfolge) als Singal oder Signalfolge.

Ein **Signal** ist eine physisch messbare Größe.

Signalcode ist wichtig für die Informationsspeicherung im Computer und der Informationsübertragung zwischen Computern.

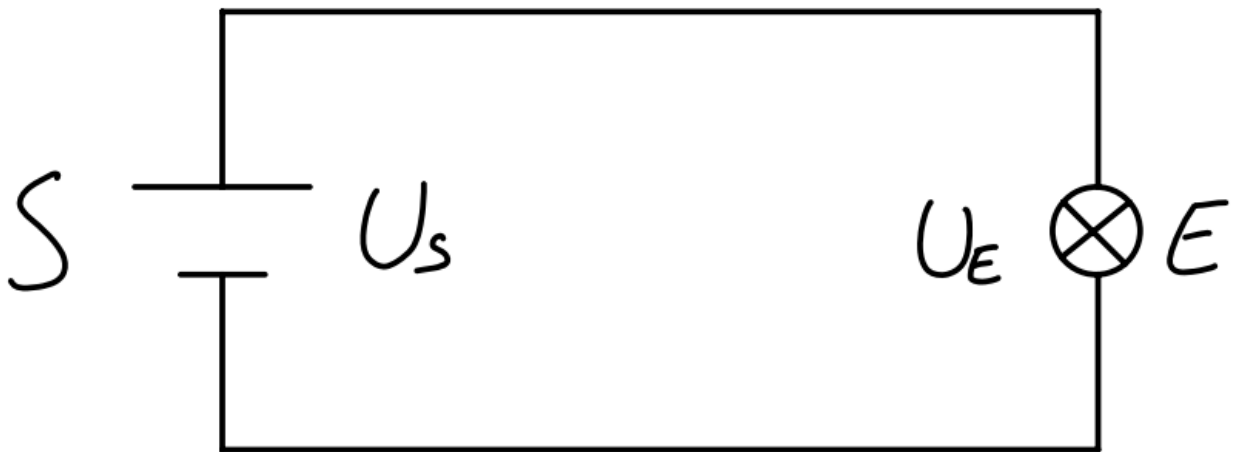
mögliche Signale:

- **mechanisch**: Kraft, Beschleunigung, Druck (Hydraulik, Pneumatik)
- **optisch**: Helligkeit (Amplitude), Farbe (Frequenz)
- **elektrisch**: bei Aufmodulation auf eine Frequenz: Frequenz, Amplitude, Spannung, Stromstärke, elektromagnetische Wellen

- **akustisch:** Lautstärke (Amplitude), Tonhöhe (Frequenz), Sprache (als komplexe Kombination der beiden vorherig genannten)

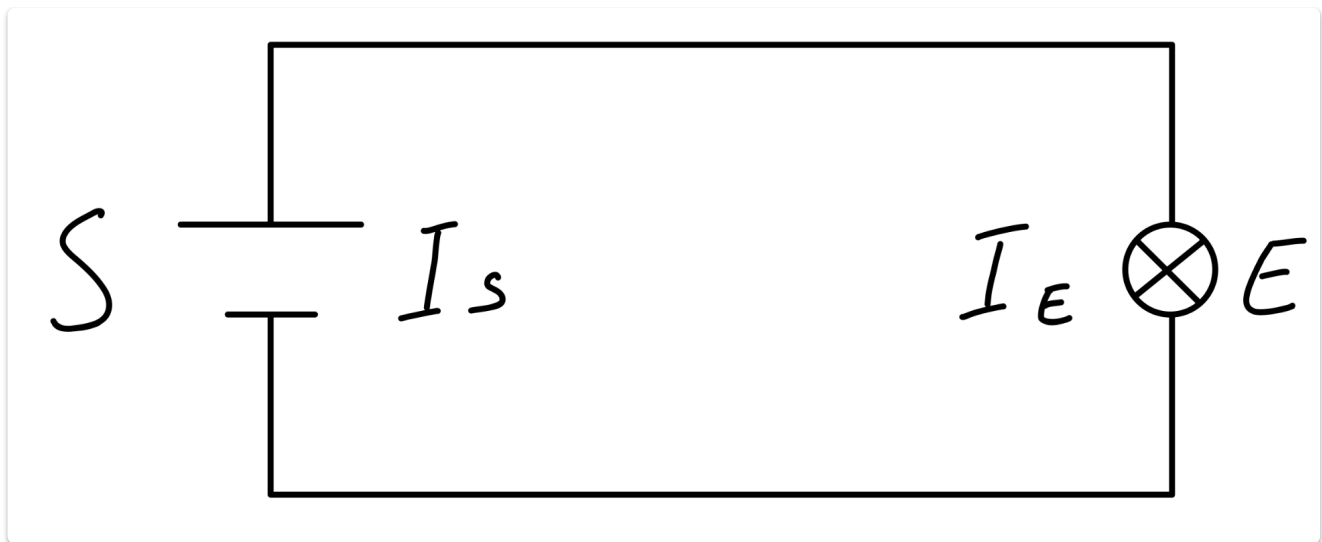
innerhalb des Rechners:

- (fast) ausschließlich elektrisch
- zwei Unterarten:
 - kabelgebunden
 - kabellos
- Strom und Spannung gehen nur kabelgebunden, bei Aufmodulation auf Trägerfrequenz geht beides
- im Rechner kann man problemlos Kabel verlegen und auf eine "Trägerfrequenz" verzichten
→ für uns kommen Strom und Spannung in Frage



$U_E < U_S$, da am Innenwiderstand des Leiters Spannungabfällt $U = R \cdot I$ (Beim Spannung Messen fließt Strom)

Außerdem kann U_E durch elektromagnetische Ströreinstrahlungen zusätzlich verstärkt und abgeschwächt werden.



$$I_E = I_S$$

Die Spannung ist abhängig vom Leitungswiderstand (Entfernung) sowie von eventuellen Störeinstrahlungen; die Stromstärke dagegen bleibt immer (nahezu) identisch.

Aber: Das Erzeugen einer bestimmten Stromstärke erfordert eine Spannung ($U = R \cdot I$). Welche bei großem Widerstand ebenfalls groß sein muss
 → damit wird viel Energie benötigt ($P = U \cdot I$; $P = \text{Leistung} = \frac{\text{Energie}}{\text{Zeit}}$)
 → außerdem dann große Wärmeentwicklung und dadurch hoher Verschleiß der elektrischen Bauteile

Im Rechner und zwischen Rechnern wird üblicherweise Spannung verwendet.

Zur Übertragung von Bit benötigt man (mindestens) zwei Spannungspegel:
 (Bsp. für "single-ended Pegel", bis dann die "0" durch "0V" dargestellt wird)

$$\text{TTL-Pegel} \begin{cases} 0 \hat{=} 0V \\ 1 \hat{=} 5V \end{cases}$$

TTL: Transistor/Transistor-Logic;

"klassisches Pegelpaar", eingeführt von *Texas Instruments* um 1960, um Interoperabilität zwischen ICs verschiedener Hersteller zu ermöglichen.

Unterschied zwischen größeren und kleineren Pegeln — Vor- und Nachteile?

- größere Sicherheit bei größeren Pegeln
- weniger Leistung (und damit geringerer Energieverlust) bei kleineren Pegeln
- kleinere Spannungsänderungen sind schneller als größere Spannungsänderungen zu erreichen
 → Heute eher kleinere Spannungspegel als *TTL-Pegel* verwendet, aber *TTL-Pegel* sind der "Klassiker" und als Beispiel oft verwendet.

Die Bitfolgen werden als Pegelfolgen dargestellt. Falls für jedes Bit die gleiche Zeit zur Verfügung steht, handelt es sich um eine "getaktete" Übertragung.

Der Takt für die Übertragung eines Bits heißt **Taktzeit** oder **Schrittzeit** oder

"Periodendauer";

Die Frequenz der Bitänderung heißt "Taktfrequenz", "Taktrate", "Schrittrate",...

Signalcodierverfahren:

- Beschreiben die zeitliche Pegelabfolge auf der Leitung für die **getaktete Übertragung** einer Bitfolge.

a) Taktrückgewinnung (TRG)

TRG ist gegeben, falls die Uhr des Empfängers auf die Uhr des Senders nur während des übertragenen Datensignals synchronisiert werden kann.

Grundvoraussetzung für TRG:

Pegelwechsel zu einem definierten Zeitpunkt

b) Gleichstromfreiheit (GSF)

Der zeitlich mittlere Pegel auf der Datensignalleitung beträgt $0V$.

- Dann kann dieser mittlere Pegel als Masse-Bezugspegel verwendet werden und die Masseleitung eingespart werden.

Grundvoraussetzung in den meisten Fällen:

symmetrische Pegel statt *Single-ended-Pegel*:

d.h. z.B.: $1 = 5V$ und $0 = -5V$

c) Störsicherheit (SSH)

Unanfälligkeit des Verfahrens gegenüber Spannungsänderungen, welche durch Störeinstrahlung von außen verursacht werden.

→ SSH ist direkt abhängig von der Anzahl der verwendeten (und zu unterscheidenden) Pegel, weil diese zum Vergleich bei allen Verfahren auf den gleichen "maximalen" Pegelhub "verteilt" werden müssen.

d) Bandbreitenbedarf (BBB)

Jeder Übertragungskanal hat nur eine beschränkte Bandbreite.

Bandbreite = obere Grenzfrequenz – untere Grenzfrequenz

Der Einfachheit halber sei untere Grenzfrequenz = $0Hz$

→ dann ist Bandbreite = obere Grenzfrequenz

Nyquist-Theorem:

Wenn mit einer bestimmten Schrittrate Daten übertragen werden sollen, wird mindestens die halbe Schrittrate als Bandbreite auf dem Übertragungskanal benötigt. Schrittrate = doppelte Bandbreite

1. Non-Return to Zero (NRZ)

Während der gesamten Schrittzeit wird der Pegel eingenommen, welcher dem übertragenem Bitwert entspricht.

Neben dem Informationssignal gibt es ein Taktsignal, welches man über eine extra Leitung übertragen kann.

Um diese Leitung einzusparen, muss man sich vorher auf eine Taktrate einigen und während der Übertragung immer wieder eine Synchronisierung der Uhren vornehmen, um ein zu weites Auseinanderlaufen der Uhren zu verhindern.

TRG bei NRZ: Bei jeder "01"-oder "10" Folge im Bitstrom. Bei längeren Folgen von nur "0" oder nur "1" ist **keine** TRG möglich.

GSF bei NRZ: bei im Datenstrom gleichverteilten "0" und "1"

Gleichverteilung im Datenstrom?

z.B. Übertragung von Texten mit UTF-8 codierte Schriftzeichen:

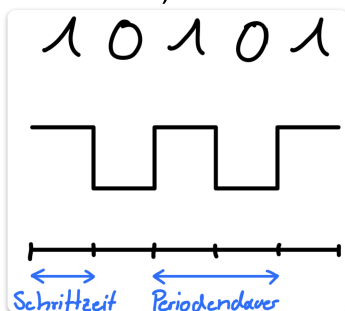
die meisten Schriftzeichen sind auch mit ASCII darstellbar, das achte Bit ist "0" → **keine** Gleichverteilung

z.B. Codierte Zeichen:

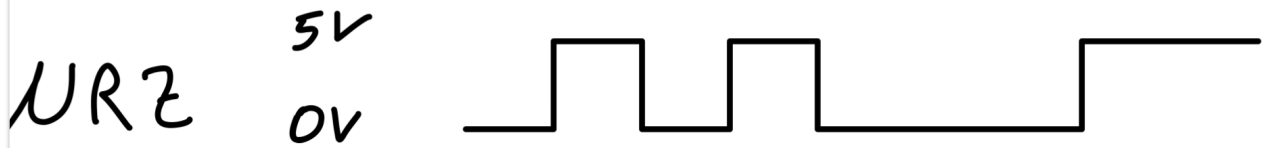
- Wertebereich wird so gewählt, dass selten große Werte übertragen werden
- höchstwertige Bit seltener "1" als "0" → **keine** Gleichverteilung
- bei verschlüsselten oder komprimierten Daten kann man im allgemeinen von einer Gleichverteilung ausgehen, bei allen anderen Daten (eher) nicht.
- eher keine GSF bei NRZ

SSH bei NRZ: Optimal, da "nur" zwei Pegel (weniger geht nicht) unterschieden können werden müssen.

BBB bei NRZ: Auf der Leitung liegt eine Frequenz, welche der halben Schrittrate entspricht, weil die Periodendauer zwei Schritzeiten entspricht, also optimal (halbe Schrittrate).



Bitfolge 010100011



2. Return-to-Zero (RZ)

Jede Schrittzeit wird in zwei gleichgroße Hälften geteilt. In der ersten Hälfte wird der Pegel eingenommen, welcher dem übertragenen Bitwert entspricht, in der zweiten Hälfte immer der 0-Pegel.

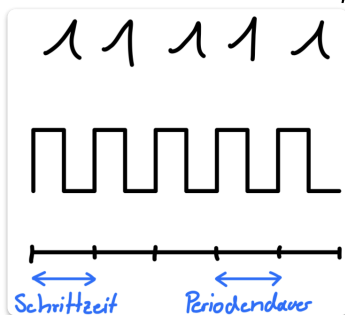
TRG bei RZ: Bei jeder "1"; nur bei längerer Folge von nur "0" gibt es **keine** TRG.

GSF bei RZ:


- unsinnig:
 - symmetrischer Pegel und nur "1"
 - single-ended-Pegel und nur "0"
- gleichverteilte "0" und "1" und "merkwürdiges" Pegelverhältnis von 3: z.B. $1 \hat{=} 15V$ und $0 \hat{=} -5V$
→ de facto NIE

TRG bei RZ: Optimal, da "nur" zwei Pegel unterschieden können werden müssen.

BBB bei NRZ: Schlecht, weil doppelt soviel wie mindestens benötigt.



Bitfolge 0 1 0 1 0 0 0 1 1

RZ $\begin{matrix} 5V \\ 0V \end{matrix}$ 

Takt 

3. Alternate Mark Inversion (AMI)

Verfahren ähnlich NRZ mit *single-ended-Pegeln*, aber jede zweite "1" wird mit dem invertierten Pegel der ersten "1" übertragen

→ insgesamt 3 Pegel: $0 \hat{=} 0V$, $1 \hat{=}$ abwechselnd $+5V$ und $-5V$

TRG bei AMI: Bei jeder "1"; denn bei jeder "1" gibt es sogar zwei definierte Pegelwechsel (nämlich zu Beginn und Ende der Schrittzeit).

GSF bei AMI: Nach jeder zweiten "1"; bei einer entsprechend langen Übertragungsdauer kann der GS-Anteil der ungeraden "1" vernachlässigt werden.

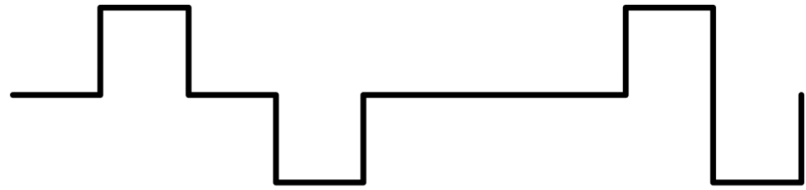
→ praktisch **immer** GSF

SSH bei AMI: Schlecht, da drei Pegel unterschieden können werden müssen.

BBB bei NRZ: halbe Schrittrate → optimal

Bitfolge 0 1 0 1 0 0 0 1 1

AMI
5V
0V
-5V



Takt



4. Manchester-Codierung

Die zu übertragenden Daten werden nicht durch einen bestimmten Pegel, sondern durch eine Pegelwechsel zu einem bestimmten Zeitpunkt (mündlich: in der "Mitte" der Schrittzeit) dargestellt.

z.B. 0-Pegel \rightarrow 1-Pegel $\hat{=}$ "1" (steigende Flanke) und 1-Pegel \rightarrow 0-Pegel $\hat{=}$ 0 (fallende Flanke)

Gegebenenfalls muss ein zusätzlicher Pegelwechsel zu Beginn der Schrittzeit eingefügt werden, dieser steht allerdings **nicht** für einen Bitwert.

TRG bei Manchester: Immer, da in der Mitte jeder Schrittzeit ein Pegelwechsel stattfindet.

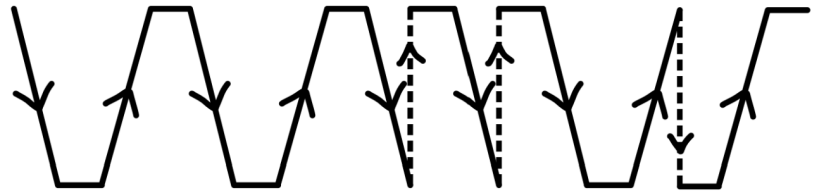
GSF bei Manchester: Immer (bei symmetrischen Pegeln), da sich die Pegel der ersten und zweiten Schrittzeit gegenseitig ausgleichen.

SSH bei Manchester: Optimal, da (nur) zwei verwendete Pegel.

BBB bei NRZ: ganze Schrittrate \rightarrow schlecht (Frequenz wird bei "00...0" oder "11...1" erzeugt)

Bitfolge 010100011

Manch. $5V$
 $0V$
 $-5V$



Takt



Übersicht

	NRZ	RZ	AMI	Manchester
TRG	$\ominus\ominus$ bei "10" oder "01"	\ominus bei jeder "1"	\ominus bei jeder "1"	\oplus immer
GSF (Grundvoraussetzung: symmetrische Pegel)	\ominus "0" und "1" gleichgestellt im Datenstrom	$\ominus\ominus$ de facto nie	\oplus nach jeder zweiten "1", praktisch immer	$\oplus\oplus$ immer
SSH (abhängig von der Anzahl verwendeter Pegel)	\oplus 2	\oplus 2	\ominus 3	\oplus 2
BBB (in Abhängigkeit der Schrittrate)	\oplus halbe Schrittrate	\ominus ganze Schrittrate	\oplus halbe Schrittrate	\ominus ganze Schrittrate

praktischer Einsatz:

Manchester: gut für Netzwerkschnittstellen, z.B. bei Ethernet eingesetzt

(Leitungseinsparung bei TRG und GSF spart Geld, SSH auf längeren Strecken ebenfalls wichtig)

NRZ: gut für "interne Schnittstellen", z.B. zwischen ICs oder verschiedenen Bauteilen
(Masse- und Taktleitung können problemlos an jedes IC geführt werden)

TRG ist bei heutigen modernen, schnellen, bit-seriellen Schnittstellen (PCI-Express, SATA)

→ Verfahren zwischen TRG, falls das Signalcodierverfahren (z.B. NRZ) von sich aus keine TRG () ermöglicht

1. *Startbitsequenz*

Vor eine feste Länge von n Nutzdatenbit wird eine Bitsequenz gestellt, die sichere TRG ermöglicht.

z.B. bei **NRZ**: "10" oder "01"

z.B. bei **RZ**: eine "1" als Startbit (keine "Sequenz")

⊕ extrem einfaches Verfahren

⊖ großer, konstanter Overhead, d.h. **Nutzdatenrate** ist immer **kleiner** als die **Schrittrate**

2. *Bitstopfen / Bit Stuffing*

Nach n gleichen Bitwerten im Nutzdatenstrom wird ein dazu **inverses Bit** als "Stopfbit" in den übertragenen **Datenstrom** eingefügt.

Bitstopfen bei $n = 3$:

0 0 1 0 1 1 1 | 1 0 0 0 | 1 1 1 | 0 Nutzdatenstrom

1. 2. 1. 1. 1. 2. 3. | 1. 1. 2. 3. | 1. 2. 3. | 1. Zähler

↑

↑

↑

0

1

0

Stopfbits

Der Empfänger muss die eingestopften Bit wieder entfernen, bevor der Nutzdatenstrom "nach oben" gegeben wird.

⊕ deutlich kleinerer Overhead und auch nur dann wenn nötig

⊖ komplexes und dadurch teures, fehleranfälliges Verfahren

⊖ Nutzdatenrate ist variabel und nicht konstant.

Einsatz z.B. bei Ethernet (nicht für TRG, aber im Bitmuster des "Frame Delimiters" im übertragenen Datenstrom auszuschließen)

3. *Blockcodierung*

Umcodierung eines n -bit-Nutzdatenblocks in einem $(n + i)$ bit langen zu übertragenden Bitblock. Aus den 2^{n+i} möglichen Blöcken werden die 2^n Blöcke ausgesucht, welche die Anforderung (hier TRG) "am besten" erfüllen.

z.B. 3B/4B-Blockcodierung; also Umsetzung eines 3-bit-Nutzdatenblocks in einen 4-bit-Block:

3 bit Blöcke		4 bit Blöcke
		0 0 0 0
		0 0 0 1
000	—	0 0 1 0
		0 0 1 1
001	—	0 1 0 0
		0 1 0 1
010	—	0 1 1 0
		0 1 1 1
		1 0 0 0
011	—	1 0 0 1
100	—	1 0 1 0
110	—	1 0 1 1
		1 1 0 0
111	—	1 1 0 1
		1 1 1 0
		1 1 1 1

- ⊕ sehr einfaches Verfahren (Realisierung über Tabelle)
- ⊕ bessere **TRG** als bei Startbitsequenz (nämlich hier z.B. zweimal in jeder Bitfolge)
- ⊕ eventuell auch Gleichstrom-Armut (wenn auch nicht perfekte **GSF**)
- ⊖ konstanter hoher Overhead (unabhängig von dessen "Notwendigkeit")

Wahl des Parameters n als Anzahl Schritte ohne **TRG**

1.	2.	3.

Im besten Fall misst der Empfänger (bei **NRZ**) den Pegel in der Mitte der angenommenen Schrittzeit, weil da der Abstand zur vorigen und nächsten Schrittzeit am größten ist.

Insgesamt dürfen (über mehrere Schritte) die Uhren von Sender und Empfänger maximal eine halbe Schrittzeit voneinander abweichen.

→ Für jede einzelne Uhr ist nur eine Abweichung von einem Viertel (also 25%) erlaubt.

Bei einer vom Uhrenhersteller spezifizierten Gangungenauigkeit von z.B. 5% wären nach 5 Schritten die 25% erreicht → $n < 5$ (z.B. $n = 4$)