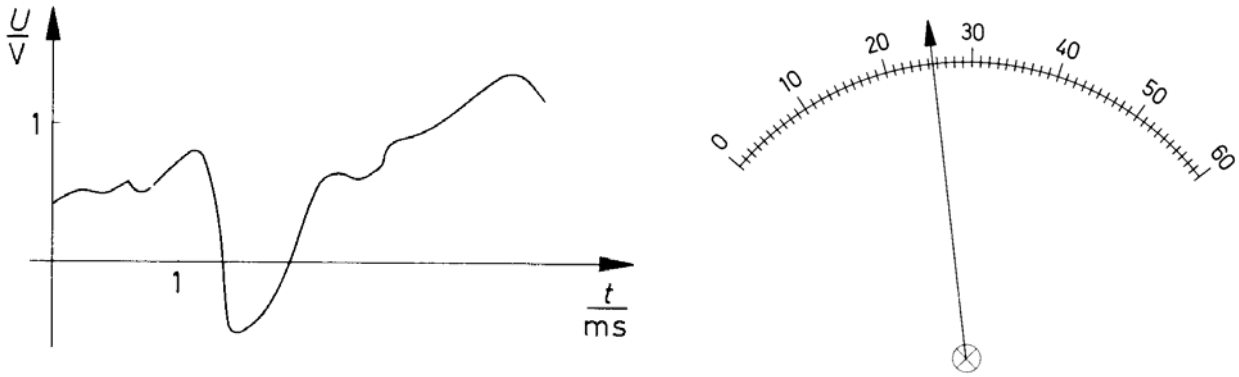
2. GRUNDLAGEN DER DIGITALEN INFORMATIONSVERARBEITUNG

2.1 Analoge und digitale Informations-darstellung

Informationen müssen in technischen Systemen mittels einer physikalischen Größe dargestellt werden. In der (modernen) Rechnertechnik sind das elektrische Größen, vorwiegend die elektrische Spannung.

Bei der analogen Größendarstellung ist die Analogiegröße proportional zu dem darzustellenden Informationswert. Sie darf in einem zulässigen Bereich jeden beliebigen Wert annehmen.



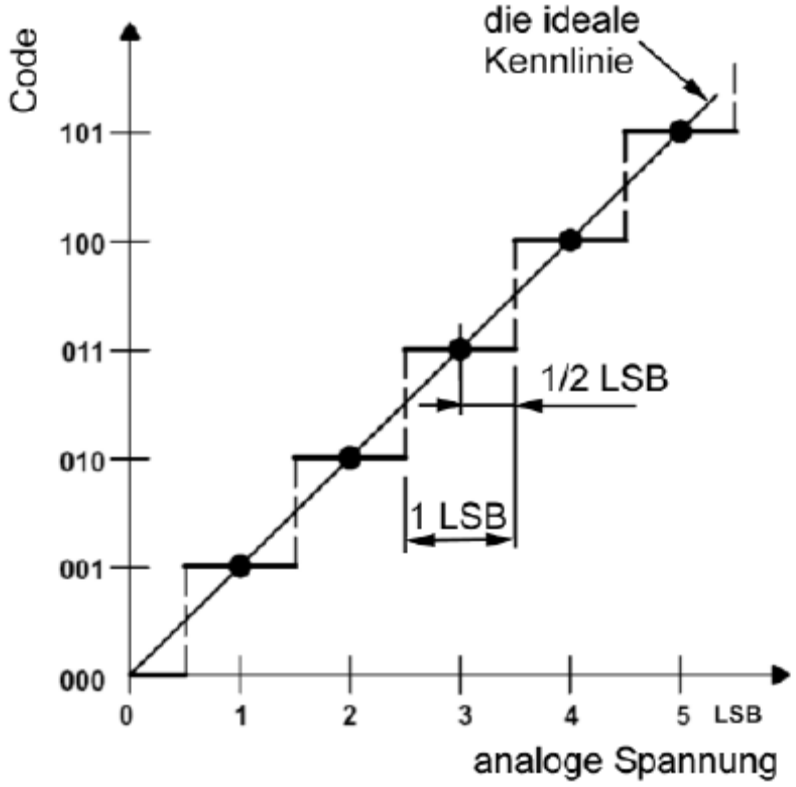
Beispiel: Zeitlicher Verlauf einer analogen Spannung und Darstellung auf einem Analog-Messgerät

Die Analogdarstellung ist sehr anschaulich und erleichtert Aussagen über Trends.

Signalwerte werden bei der Analogverarbeitung durch viele (physikalische) Effekte, z.B. Rauschen etc., verfälscht. Die erreichbare Genauigkeit hängt somit von der Genauigkeit der Messung und Verarbeitung ab.

Die erreichbare Genauigkeit liegt in der Praxis im Prozent- und Promillebereich. Eine höhere Genauigkeit erfordert einen sehr (zu) hohen technischen Aufwand und stößt an Grenzen der physikalischen Realisierbarkeit.

Die digitalen Größendarstellung (digitus (lat.): der Finger) basiert auf abzählbaren Elementen. D.h. die darzustellende Information wird in Stufen repräsentiert.



Jeder Stufe wird eine Zahl zugeordnet, die dann als informationstragender Parameter verarbeitet wird.

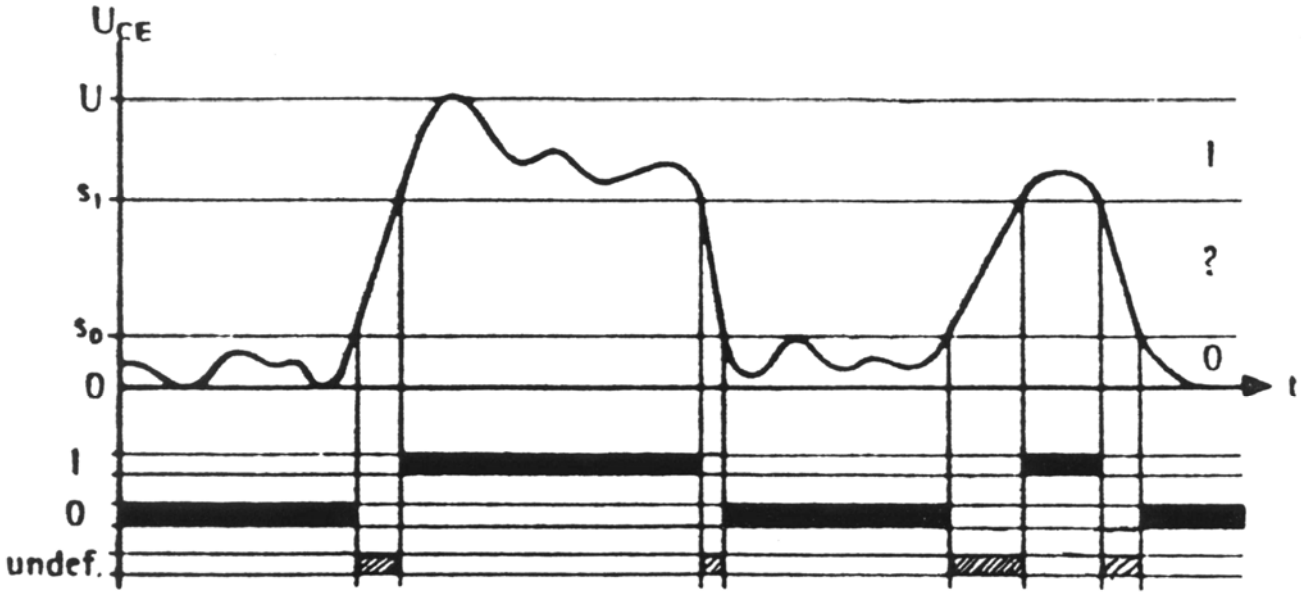
Nach der Digitalisierung hängt die Genauigkeit der Verarbeitung nicht mehr von physikalischen Effekten ab und kann durch die Erhöhung der Stufenzahl beliebig gesteigert werden.

Digitale Größen werden zu ihrer Verarbeitung auf Signale abgebildet, die zwei-, drei- oder mehrwertig sein können, also einen von zwei, drei oder mehr Zuständen annehmen.

Üblicherweise werden in heutigen technischen Systemen zweiwertige Signale verwendet, also die binäre Darstellung. Dabei werden die Zustände ´0´ und ´1´ (bzw. ´true´ und ´false´) unterschieden.

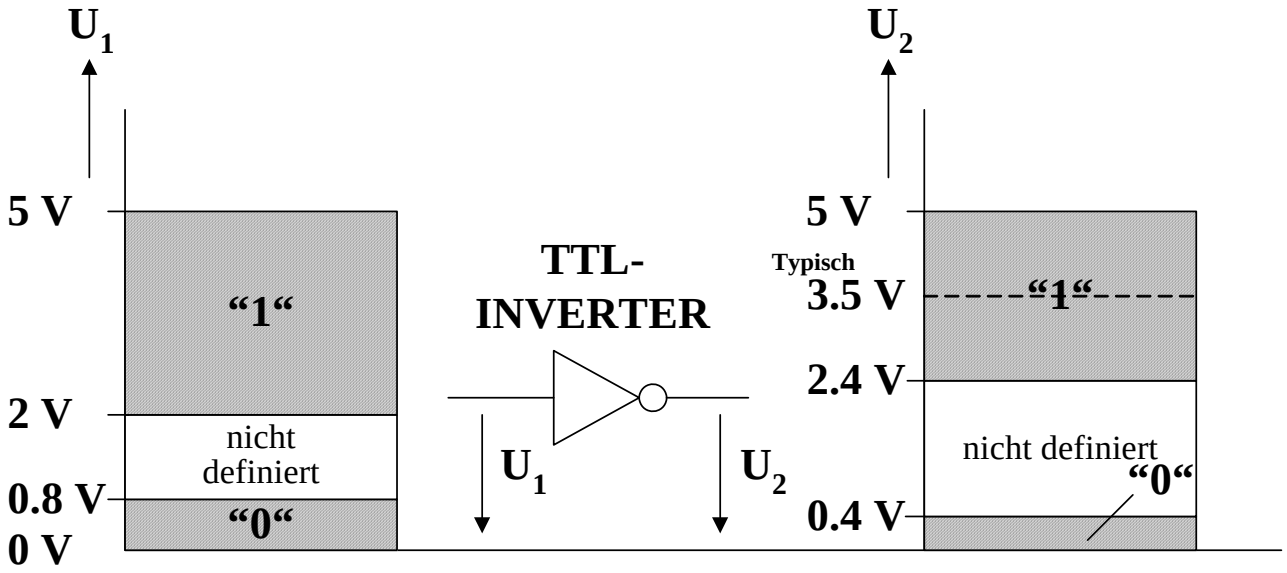
Beachte: In elektronischen Systemen werden technologisch bedingt auch diese binären Signale durch analoge Größen repräsentiert und unterliegen deshalb den gleichen Störungen wie analoge Größen.

Definition von 0 und 1:
Damit Störungen besser toleriert werden, werden Toleranzbänder für ´0´ und ´1´ definiert und mit einem Schaltabstand (´Störspannungsabstand´) versehen.
So entstehen durch die Anstiegs- und Abfallzeiten *undefinierte* (Übergangs-)Zustände, die nicht eindeutig 0 oder 1 zugeordnet werden können (⇒ bei Gatterlaufzeit berücksichtigen).



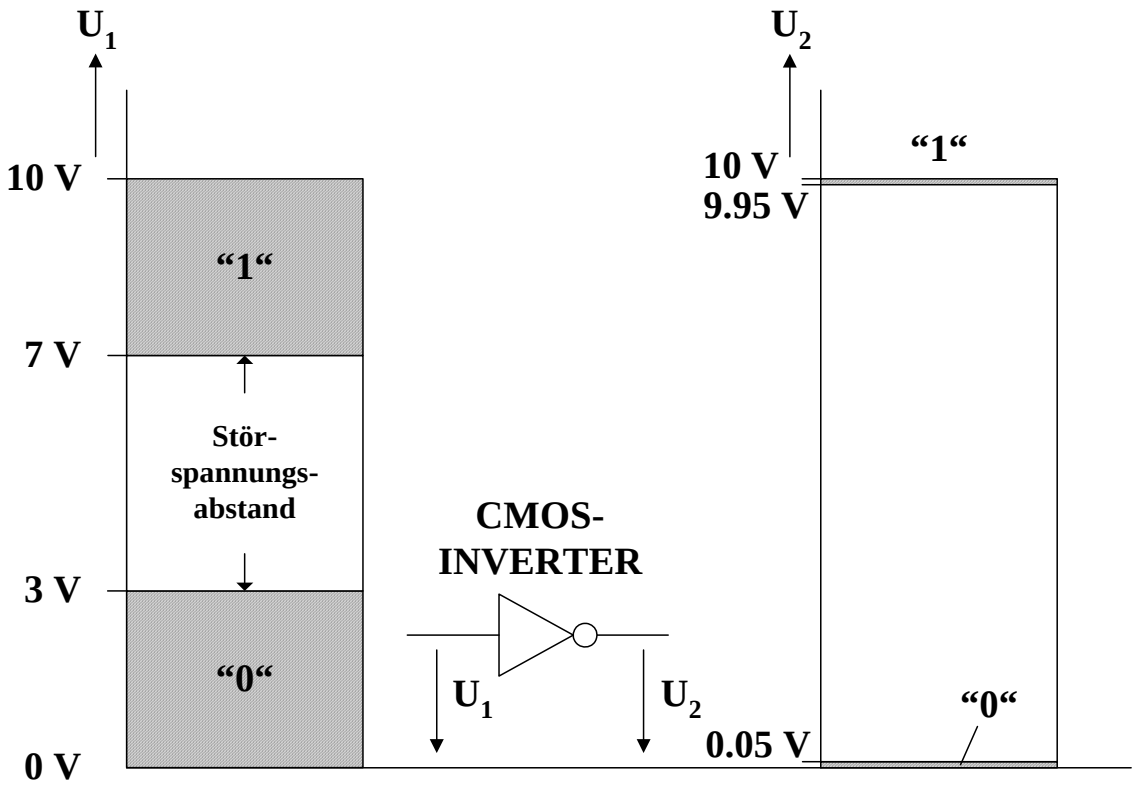
Definition von Spannungspegel

Beispiel: TTL-Pegel



I. Allg. unterschiedliche Toleranzbereiche für Ein- und Ausgang.

Beispiel für CMOS-Logikpegel:



Zuordnung von Spannungspegeln zu logischen Zuständen

Die Zuordnung der binären Zustände ´0´ und ´1´ zu den Spannungspegeln ist eine Definitionssache.

Positive Logik:
Dem positiveren von zwei definierten Spannungspegeln wird die logische ‘1‘, dem negativeren die ‘0‘ zugeordnet.

Negative Logik:
Dem negativeren von zwei definierten Spannungspegeln wird die logische ‘1‘, dem positiveren die ‘0‘ zugeordnet.

Im Folgenden: nur **positive Logik** betrachtet

2.2 Zahlensysteme

Um mit Zahlen (mathematische) Operationen auszuführen wie Addition, Subtraktion, Multiplikation, müssen sie in einer geeigneten Darstellung vorliegen. Dazu wird bei technischen Systemen ein Stellenwertsystem (polyadisches System) verwendet, gekennzeichnet durch die Basis $B \geq 2$.

Eine Zahl Z ergibt sich dann in der allgemein gültigen Form:

$$Z = c_{n-1} \cdot B^{n-1} + c_{n-2} \cdot B^{n-2} + ... + c_1 \cdot B^1 + c_0 \cdot B^0 + c_{-1} \cdot B^{-1} + ... + c_{-m} \cdot B^{-m}$$

oder verkürzt: $$\boxed{Z = \sum_{i=-m}^{n-1} c_i \cdot B^i}$$

oder: $$\boxed{Z = c_{n-1} c_{n-2} \ldots c_1 c_0 . c_{-1} \ldots c_{-m}}$$

mit:
- B : Basis
- c_i : Koeffizient
- i : Ordnungszahl
- n : Stellenzahl links vom Komma
- m : Stellenzahl rechts vom Komma

Für die Koeffizienten gilt: $0 \leq c_i \leq B - 1$

In einem polyadischen Zahlensystem können mit n Stellen und der Basis B $N=B^n$ verschiedene Zahlen dargestellt werden.

In der Rechnertechnik werden das Dual- (Basis B=2), das Oktal- (B=8) und das Hexadezimalzahlensystem (B=16) angewendet.

Dualzahlen (Binärzahlen)

Von besonderer Bedeutung ist das duale (binäre) Zahlensystem, weil

- eine Dualstelle (´Bit´) technisch leicht realisiert und verarbeitet werden kann (z.B. Schalter, Relais, Transistor),
- die Verarbeitung relativ störunempfindlich ist,
- Schaltungen zur Verarbeitung von Dualzahlen leicht mit Hilfe der Schaltalgebra zu entwerfen sind.

Durch die Basis 2 sind bei Dualzahlen nur die Koeffizienten

$$c_i \in \{0,1\}$$

zulässig.

Eine ganzzahlige Dualzahl *Z* kann bei *n* Stellen Zahlen bis 2^n-1 (einschließlich 0) darstellen.

$$Z = c_{n-1} \cdot 2^{n-1} + c_{n-2} \cdot 2^{n-2} + ... + c_1 \cdot 2^1 + c_0 \cdot 2^0$$

Die Gewichtung erfolgt je nach Stellenwert mit:

n	***2^n***		***n***	***2^n***		***n***	***2^n***
0	1		8	256		16	65.536
1	2		9	512		17	131.072
2	4		10	1024		18	262.144
3	8		11	2048		20	1.048.576
4	16		12	4096		24	16.777.216
5	32		13	8192		32	4.294.967.296
6	64		14	16384		64	$1{,}846...\cdot 10^{19}$
7	128		15	32768		128	$3{,}402\ldots\cdot 10^{38}$

Hinweise:

Dualzahlen werden oft auch als Bit-String betrachtet.

Das ganz links stehende Bit wird meist als **MSB** (most significant bit) und das ganz rechts stehende Bit als **LSB** (least significant bit) bezeichnet.

Bei nahezu allen modernen Prozessoren werden Wortbreiten verwendet, die ganzzahlige Vielfache von 8 sind (8, 16, 32, 64). Die dadurch darstellbaren Zahlenbereiche entsprechen i.d.R. dem Wertebereich der Integerzahlen des Prozessors.

Bei Betrachtung der Hardwareseite werden führende Nullen mit dargestellt, um die ganze Wortbreite auszufüllen.

Beispiel: $537_{10} = (0000\ 0010\ 0001\ 1001)_2 = 0219_{16}$

Die in der Computertechnik verwendeten Einheitenvorsätze K (kilo), M (mega), G (giga) und T (tera) beziehen sich (im Gegensatz zu den Einheitenvorsätze im Dezimalsystem) auf die Zweierpotenzen 2^{10}, 2^{20}, 2^{30} und 2^{40}.

Beispiele: $4\ K = 4 \cdot 2^{10} = 4 \cdot 1024 = 4096$

$16\ M = 16 \cdot 2^{20} = 16 \cdot 1.048.576 = 16.777.216$

$1\ T = 1 \cdot 2^{40} = 1{,}0995\ldots \cdot 10^{12}$

Oktal- und Hexadezimalzahlen

Oktal- (Basis 8 = 2^3) und Hexadezimalzahlen (Basis 16 = 2^4) werden häufig für eine kompaktere Darstellung von Dualzahlen verwendet. (Die 4 Bit einer Hexadezimalzahl werden auch ´Nibble´ genannt.)

dezimal (Basis 10)	**dual / binär** (Basis 2)	**oktal** (Basis 8)	**hexadezimal** (Basis 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Oktal- und Hexadezimalzahlen fassen jeweils 3 bzw. 4 Stellen einer Dualzahl zu einer Ziffer zusammen. Die Umwandlung ist daher sehr einfach.

Beispiele: $(010\ 110\ 001\ 101, 111\ 100\ 000\ 110)_2 = (26153,7406)_8$

$(0101\ 1000\ 1101, 1111\ 0000\ 0110)_2 = (58D,F06)_{16}$

$(673,12)_8 = (110\ 111\ 011, 001\ 010)_2$

$(3A6,C)_{16} = (0011\ 1010\ 0110, 1100)_2$

2.3 Umwandlung von Zahlendarstellungen

Umwandlung von und in Dezimalzahlen

Bei der Umwandlung von Zahlen in eine andere Basis werden Vor- und Nachkommastellen separat behandelt.

Die Umwandlung einer Dezimalzahl *Z,z* in eine Zahl mit einer anderen Basis *B* basiert auf der rekursiven Division (Vorkommastellen) durch bzw. Multiplikation (Nachkommastellen) mit der (stellengewichteten) Basis *B* der Zieldarstellung.

$$Z,z \rightarrow c_{n-1}c_{n-2}\ldots c_1c_0,c_{-1}c_{-(m-1)}c_{-m}$$

Vorkommastellen		Nachkommastellen
$Z / B = S_1$	Rest c_0	$z \cdot B = c_{-1} + s_1$
$S_1 / B = S_2$	Rest c_1	$s_1 \cdot B = c_{-2} + s_2$
$S_2 / B = S_3$	Rest c_2	$s_2 \cdot B = c_{-3} + s_3$
.		.
.		.
$S_{n-2} / B = S_{n-1}$	Rest c_{n-2}	$s_{m-2} \cdot B = c_{-(m-1)} + s_{m-1}$
$S_{n-1} / B = \mathbf{0}$	Rest c_{n-1}	$s_{m-1} \cdot B = c_{-m} + \mathbf{0}$

Beispiel: Umwandlung von 109,78125 in eine Dualzahl

109 / 2 = 54	Rest **1** (LSB)	0,78125 · 2 = **1** + 0,5625
54 / 2 = 27	Rest **0**	0,5625 · 2 = **1** + 0,125
27 / 2 = 13	Rest **1**	0,125 · 2 = **0** + 0,25
13 / 2 = 6	Rest **1**	0,25 · 2 = **0** +0,5
6 / 2 = 3	Rest **0**	0,5 · 2 = **1** + 0,0
3 / 2 = 1	Rest **1**	
1 / 2 = 0	Rest **1** (MSB)	

$$109{,}78125_{10} = 1101101{,}11001_2$$

Umwandlung von 109,78125 in eine Oktalzahl

109 / 8 = 13	Rest **5**	0,78125 · 8 = 0,25 +	**6**
13 / 8 = 1	Rest **5**	0,25 · 8 = 0,0 +	**2**
1 / 8 = 0	Rest **1**		

$$109{,}78125_{10} = 155{,}62_{8}$$

Bei der Umwandlung von Zahlen mit beliebiger Basis in Dezimalzahlen können Vor- und Nachkommastellen analog rekursiv nach folgendem Berechnungsschema umgewandelt werden:

Vorkommastellen	Nachkommastellen
$c_{n-1} = S_1$	$c_{-m} = s_1$
$S_1 \cdot B + c_{n-2} = S_2$	$s_1/B + c_{-(m-1)} = s_2$
$S_2 \cdot B + c_{n-3} = S_3$	$s_2/B + c_{-(m-2)} = s_3$
.	.
.	.
$S_{n-2} \cdot B + c_1 = S_{n-1}$	$s_{m-1}/B + c_{-1} = s_m$
$S_{n-1} \cdot B + c_0 = Z$	$s_m/B = z$

Beispiel: Dualzahl 111000,011

1 = 1	1 = 1
1·2 + 1 = 3	1/2 + 1 = 1,5
3·2 + 1 = 7	1,5/2 + 0 = 0,75
7·2 + 0 = 14	0,75/2 + 0 = **0,375**
14·2 + 0 = 28	
28·2 + 0 = **56**	

$$111000{,}011_2 = 1\cdot2^5+1\cdot2^4+1\cdot2^3+0\cdot2^2+0\cdot2^1+0\cdot2^0+0\cdot2^{-1}+1\cdot2^{-2}+1\cdot2^{-3}$$
$$= 32 + 16 + 8 + 0{,}25 + 0{,}125$$
$$= 56{,}375_{10}$$

Oktalzahl 374,24

3 = 3	4 = 4
3·8 + 7 = 31	4/8 + 2 = 2,5
31·8 + 4 = **252**	2,5/8 + 0 = **0,3125**

$$374{,}24_8 = 3\cdot8^2 + 7\cdot8^1 + 4\cdot8^0 + 2\cdot8^{-1} + 4\cdot8^{-2}$$
$$= 192 + 56 + 4 + 0{,}25 + 0{,}0625$$
$$= 252{,}3125_{10}$$

2.4 Darstellung negativer Dualzahlen

In der Mathematik werden positive und negative Zahlen normalerweise durch ihren Betrag und ihr Vorzeichen dargestellt.

In der Digitaltechnik werden bei der Vorzeichen-Betrags-Darstellung (sign magnitude) von Dualzahlen normalerweise das Vorzeichen ´+´ durch eine ´0´ und negative Zahlen durch ´1´ repräsentiert.

Bei dieser Darstellung sind die Ziffern (Bits) für die Zahlen und Vorzeichen nicht per se unterscheidbar. Die Unterscheidung erfolgt durch die Stelle, wobei üblicherweise das Vorzeichenbit ganz links steht.

Für eine Hardwareimplementierung ist diese Vorzeichen-Betrags-Darstellung nicht immer optimal. Hier wird oft die Komplementdarstellung bevorzugt.

Das Einer-Komplement einer Dualzahl erhält man durch einfache Invertierung, also das Austauschen der Nullen und Einsen. Das Zweier-Komplement erhält man aus dem Einerkomplement durch Addition einer 1.

Eine andere Art, negative Zahlen mit Dualzahlen darzustellen, ist die Addition eines festen Offsets, meist eine Konstante mit der Wertigkeit der höchsten Bitstelle (MSB). D.h. die Null wird um den halben Wertebereich verschoben. Positive Zahlen beginnen dann mit einer ´1´, negative mit einer ´0´. Diese Darstellung wird häufig an der Schnittstelle zu analogen Signalen (Analog-Digital- und Digital-Analog-Umsetzer) verwendet.

Dezimal-zahl	Vorzeichen-Betrags-Darstellung	Einer-Komplement-Darstellung	Zweier-Komplement-Darstellung	Offset-binäre Darstellung
+ 7	0111	0111	0111	1111
+ 6	0110	0110	0110	1110
+ 5	0101	0101	0101	1101
+ 4	0100	0100	0100	1100
+ 3	0011	0011	0011	1011
+ 2	0010	0010	0010	1010
+ 1	0001	0001	0001	1001
+ 0	0000	0000	0000	1000
- 0	1000	1111	0000	1000
- 1	1001	1110	1111	0111
- 2	1010	1101	1110	0110
- 3	1011	1100	1101	0101
- 4	1100	1011	1100	0100
- 5	1101	1010	1011	0011
- 6	1110	1001	1010	0010
- 7	1111	1000	1001	0001
- 8	-	-	1000	0000

Anmerkungen: Die +0 und -0 unterscheiden sich bei der Vorzeichen-Betrags- und der Einer-Komplement-Darstellung, bei der Zweier-Komplement- und der offset-binären Darstellung aber nicht.

Die Zweier-Komplementdarstellung und die offset-binäre Darstellung erfassen daher eine Zahl mehr als die anderen.

2.5 Rechnen mit Dualzahlen

Addition

Die Addition von Dualzahlen erfolgt stellengewichtet.

Der Ablauf ist also wie der bei Dezimalzahlen, außer dass häufiger Überträge auftreten, weil die einzelnen Stellen nur die Werte ´0´ und ´1´ annehmen können.

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 Übertrag 1
(auf die nächst höherwertige Stelle)

Beispiel: $13_{10} + 11_{10}$

13_{10} = 1101
11_{10} = 1011
Überträge 1111
11000 = 24_{10}

Subtraktion

Die Subtraktion zweier Dualzahlen erfolgt meist durch die Addition des Zweier-Komplements (zur nächst höheren Basispotenz). Hierbei berechnet sich die Differenz *D* aus Minuend *M* und Subtrahend *S* wie folgt:

$$D = M - S = M + (B^n - S) - B^n = M + K - B^n$$

mit: $K = B^n - S$ (Zweierkomplement)

Beispiel: 123 - 11 = 112

$123_{10} = 1111011_2$; $11_{dez} = 1011_2$

$$\begin{aligned} 1111011 - 1011 &= 1111011 + (10000000 - 1011) - 10000000 \\ &= 1111011 + (1111111 - 1011 + 1) - 10000000 \\ &= 1111011 + (1110100 + 1) - 10000000 \\ &= 1111011 + 1110101 - 10000000 \\ &= 11110000 - 10000000 \\ &= 1110000 \end{aligned}$$

$1110000_2 = 112_{10}$

Multiplikation

Die Multiplikation von Dualzahlen wird i.d.R. wie bei Dezimalzahlen auf eine stellenbewertete Addition zurückgeführt.

Das ist einfacher als bei Dezimalzahlen zu realisieren, weil für jede ´1´ im Multiplikator lediglich der Multiplikand einfach angesetzt, also addiert werden muss, sonst nicht, d.h. 0.

Bei der Multiplikation zweier Dualzahlen ist zu berücksichtigen, dass die Stellenzahl des Produkts der Summe der Stellenzahlen von Multiplikator und Multiplikand entsprechen kann, sich also die Stellenzahlen addieren können.

Beispiel: $215 \cdot 35$

$215_{10} = 11011001_2$; $35_{10} = 100011_2$

```
11011001 · 100011
        11011001
       11011001
      00000000
     00000000
    00000000
   11011001
   1110110101011
```

$1110110101011_2 = 7595_{10}$

Beispielablauf für Multiplikation im Zweierkomplement

Step	Action	F	Accumulator A	Register Q	
0	Initialize registers	0	00000000	10110011	= multiplier *X*
1			11010101		= multiplicand *Y* = M
	Add M to A	1	11010101	10110011	add if LSB of Q = 1
	Shift A.Q	1	11101010	11011001	right shift
2			11010101		
	Add M to A	1	10111111	11011001	add if LSB of Q = 1
	Shift A.Q	1	11011111	11101100	right shift
3			00000000		
	Add zero to A	1	11011111	11101100	add if LSB of Q = 1
	Shift A.Q	1	11101111	11110110	right shift
4			00000000		
	Add zero to A	1	11101111	11110110	add if LSB of Q = 1
	Shift A.Q	1	11110111	11111011	right shift
5			11010101		
	Add M to A	1	11001100	11111011	add if LSB of Q = 1
	Shift A.Q	1	11100110	01111101	right shift
6			11010101		
	Add M to A	1	10111011	01111101	add if LSB of Q = 1
	Shift A.Q	1	11011101	10111110	right shift
7			00000000		
	Add zero to A	1	11011101	10111110	add if LSB of Q = 1
	Shift A.Q	1	11101110	11011111	right shift
8			11010101		
	Subtract M from A	1	00011001	11011111	correct add of sign
	Set LSB of Q to 0		00011001	11011110	= product *P= A.Q*

Hinweis: Register A und Q sind konkateniert, d.h. verkettet (A.Q). Sie enthalten schließlich das Produkt.

Das Vorzeichen von M wird über das F-Register gehalten und nach A.Q geschoben.

Korrektur im Schritt 8 abhängig von Vorzeichen.

Binäres Multiplizieren für Festpunktzahlen

Zahlendarstellung:

- Beispiel: 8 Bit Wortbreite
- negative Zahlen mit Betrag und Vorzeichen (Sign Magnitude-Darstellung)
- Betrag wird als echter Dualbruch dargestellt (d. h. Komma vor der ersten Stelle):

$$X = (x_0.x_1\, x_2 \ldots x_7)_2$$

ergibt:

$$N = (-1)^{x_0} \cdot (\sum_{i=1}^{7} x_i \cdot 2^{-i})$$

Multiplikation $P \leftarrow X \times Y$

- Vorzeichen:

$$p_0 \leftarrow x_0 \text{ XOR } y_0$$

- Produkt der Beträge (durch Addition und Schieben):

Addition (abhängig von Bit x_{7-i}):

$$P_i \leftarrow P_i + x_{7-i} \cdot Y$$

Rechtsshift: $P_{i+1} \leftarrow 2^{-1} \cdot P_i$

mit $P_0 = 0$, $P_7 = P$ und $i = 0, \ldots, 6$

Binäre Multiplikation mit negativen Zahlen

Betrag und Vorzeichen

- Umwandlung von Komplementdarstellung in Darstellung mit Betrag und Vorzeichen
- Multiplikation der Beträge, Ermittlung des Vorzeichens gemäß $p_0 \leftarrow x_0$ XOR y_0
- Bei negativem Vorzeichen des Ergebnisses Rückwandlung in Komplementdarstellung

Nachteil: Zusätzlicher Zeitaufwand für Komplementbildung(en)

Festpunkt-Multiplikation direkt im Zweierkomplement

Beispiel: Verfahren nach Robertson

Festpunktdarstellung mit Dezimalpunkt vor 1. gültiger Stelle x_1, Vorzeichen in x_0:

$$-X = \overline{x}_0 . \overline{x}_1 \overline{x}_2 \ldots \overline{x}_{n-2} \overline{x}_{n-1} + 0.00\ldots01$$

mit $\overline{x}_i = 1 - x_i$ (modulo 2)

Damit wird

$$\begin{aligned} -X &= (1.11\ldots1)_2 - (x_0.x_1x_2\ldots x_{n-2}x_{n-1})_2 + (0.00\ldots01)_2 \\ &= (10.00\ldots0)_2 - (x_0.x_1x_2\ldots x_{n-2}x_{n-1})_2 \\ &= 2 - X \quad (\text{modulo} 2) \end{aligned}$$

Für positive X ($x_0 = 0$) gilt:

$$X = \sum_{i=0}^{n-1} 2^{-i} \cdot x_i = \sum_{i=1}^{n-1} 2^{-i} \cdot x_i$$

Für negative X ($x_0 = 1$) gilt:

$$\begin{aligned} -X &= (10.00\ldots0)_2 - ((0.x_1x_2\ldots x_{n-2}x_{n-1})_2 + (1.00\ldots0)_2) \\ &= (1.00\ldots0)_2 - (0.x_1x_2\ldots x_{n-2}x_{n-1})_2 \\ &= 1 - \sum_{i=1}^{n-1} 2^{-i} \cdot x_i \end{aligned}$$

$\Rightarrow$ (allgemein für diese Festpunktdarstellung)

$$X = -2^0 \cdot x_0 + \sum_{i=1}^{n-1} 2^{-i} \cdot x_i$$

<u>Beispiel:</u> n = 4 X = $(1.011)_2$

$$X = -2^0 \cdot 1 + 2^{-1} \cdot 0 + 2^{-2} \cdot 1 + 2^{-3} \cdot 1$$

$$= -0.625$$

(1) X und Y positiv ($x_0 = y_0 = 0$)

Normale Multiplikation mit Schieben und Addieren. Vorzeichen in y_0 wird genauso wie die übrigen Stellen behandelt:

$P_i \leftarrow P_i + x_{7-i} \cdot Y;\ \ P_{i+1} \leftarrow 2^{-1} \cdot P_i,\ \ i = 0,...,6;\ \ P_0 = 0$

(2) X positiv, Y negativ ($x_0 = 0, y_0 = 1$)

Rechtsshift von P_i mit Nachziehen von 0 liefert ***nicht*** mehr die gewünschte Division durch 2.

<u>Beispiel:</u>

$(1.110...0)_2 = -1 \bullet 2^0 + 1 \bullet 2^{-1} + 1 \bullet 2^{-2} = -0.25$

nach Shift *positive* Zahl:

$(0.1110...0)_2 = 1 \bullet 2^{-1} + 1 \bullet 2^{-2} + 1 \bullet 2^{-3} = 0.875$

<u>Korrektur:</u> ("arithmetischer Shift")

Nachziehen von 1 statt 0 (entspr. Addition von -1)

$(1.1110...0)_2 = -1 \bullet 2^0 + 1 \bullet 2^{-1} + 1 \bullet 2^{-2} + 1 \bullet 2^{-3}$
$= -0.125$

<u>Realisierung:</u>

Flipflop F (s. unten) zur Bitspeicherung (Anfangswert 0) mit

$$A(0) = F \leftarrow y_0 \cdot x_{7-i} \vee F$$

Wird für $y_0 = 1$ bei erstem $x_{7-i} = 1$ (d. h. erster 'echter' Addition) gesetzt und bleibt dann 1.

(3) X negativ, Y positiv ($x_0 = 1$, $y_0 = 0$)

Nach 7 Schiebe/Addier-Schritten ist

$$P_7 = \sum_{i=1}^{7} 2^{-i} \cdot x_i \cdot Y$$

Korrektur: Subtraktion von Y im letzten Schritt

$$P \leftarrow P_7 - Y$$

Damit ist:

$$P = -Y + \sum_{i=1}^{7} 2^{-i} \cdot x_i \cdot Y = (-x_0 \cdot 2^0 + \sum_{i=1}^{7} 2^{-i} \cdot x_i) \cdot Y$$
$$= X \cdot Y$$

mit X im Zweierkomplement

(4) X und Y negativ ($x_0 = y_0 = 1$)

Kombination der Korrekturen aus (2) und (3).

Division

Die Division von Dualzahlen erfolgt wie bei Dezimalzahlen über fortlaufende (stellengewichtete) Subtraktion.
Sie ist ebenfalls einfacher als bei Dezimalzahlen, weil nur einfache Werte des Divisors abgezogen werden.

Die Subtraktion wird i.d.R. wieder durch eine Addition des Komplements (zur nächst höheren Basispotenz) durchgeführt. Wenn die Addition des Komplements **keinen** Übertrag ergibt, ist das entsprechende Teilergebnis 0 und es muss dann mit dem alten Minuenden plus der nächsten Stelle des Dividenden weitergerechnet werden.

Beispiel: 667 : 29

$667_{10} = 1010011011_2$;

$29_{10} = 11101_2$; $-29 = 00011_{\text{2er-Kompl}}$

```
 1010011011 : 11101 = 010111
 00011
0 10111
 101001
  00011
 1 011001
   00011
  0 11100
    110010
     00011
    1 101011
      00011
     1 011101
       00011
      1 00000
```

$010111_2 = 23_{10}$

Realisierung der Division

Verschiedene Realisierungen, z. B. analog zur Multiplikation durch wiederholte stellengewichtete Subtraktion

$\rightarrow$ Ablaufsteuerung nötig

Probleme:

(1) Ergebnis ist Quotient q und Rest r

$$a : b = q \text{ Rest } r \quad \text{mit} \quad a = q \cdot b + r$$

(2) Bei der Multiplikation mittels Multiplikatorbit ist vorher feststellbar, ob addiert werden soll oder nicht.

Bei der Division ist erst nach der Subtraktion am Vorzeichenwechsel des Dividenden erkennbar, ob die Subtraktion erforderlich war (d. h. Subtraktion evtl. rückgängig machen, z. B. alten Wert aufheben).

(3) Der Quotient muss ins Quotientenregister passen.

(4) Vorzeichenbehandlung analog zur Multiplikation (oft Betrag und Vorzeichen).

(5) Fehlermeldung bei Division durch Null.

Beispiel: 5 : 2 = 2 Rest 1; N = 4

0101 : 0010 = 0010
010
0001 Rest

Ablauf mit doppelt langem Dividenden Q ← A.Q 'div' B

		B	1110				Zweierkomplement von Divisor
C	0	A	0000	—	0101	Q	Dividend im Register A.Q
C	0	A	0000	—	1010	Q	Linksshift 1
C	0	A	1110	—	1010	Q	Addition 1
C	0	A	0000	—	1010	Q	Da Vorzeichenwechsel: Addition rückgängig machen
C	0	A	0001	—	0100	Q	Linksshift 2
C	0	A	1111	—	0100	Q	Addition 2
C	0	A	0001	—	0100	Q	Da Vorzeichenwechsel: Addition rückgängig machen
C	0	A	0010	—	1000	Q	Linksshift 3
C	0	A	0000	—	1000	Q	Addition 3
C	0	A	0000	—	1001	Q	Kein Vorzeichenwechsel: QN← 1
C	0	A	0001	—	0010	Q	Linksshift 4
C	0	A	1111	—	0010	Q	Addition 4
C	0	A	0001	—	0010	Q	Da Vorzeichenwechsel: Addition rückgängig machen
			Rest		Quotient		

Voraussetzung: B > A (oberer Teil des Dividenden größer Divisor), sonst C = 1 (Quotient passt nicht in Register Q) und Abbruch!

Ablaufsteuerung führt genau N = 4 Subtraktions-/Schiebe-Schritte durch.

2.6 Gleitkommazahlen

Um größere Wertebereiche abzudecken und/oder reelle Zahlen (zumindest mit hinreichender Genauigkeit) digital mit handhabbarer Bitstellenzahl darstellen zu können, werden bei Computern Gleitkommazahlen (Fließpunkt, *Floating Point*) mit der entsprechenden Arithmetik verwendet.

Beispiele:

$0{,}000000001\ s = 1 \cdot 10^{-9}\ s = 1\ ns$

$3.155.760.000 = 3{,}15576 \cdot 10^{9}$
(Sekunden in einem Jahrhundert
→ nicht mehr mit signed 32-Bit-Integer darstellbar)

$\pi = 3{,}14159265\ldots$

$e = 2{,}712828\ldots$

Ansatz:

Exponential-Darstellung mit Mantisse und Exponent als Dualzahlen fester Länge,

ggf. mit Vorzeichenbit (*sign magnitude*-Darstellung)

Vorgehen:

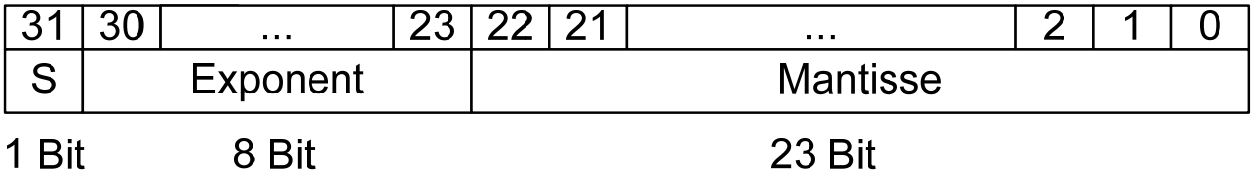
Normalisierung der Zahl so, dass eine Eins oder Null vor dem Komma steht (die dann bei der digitalen Darstellung weggelassen werden können)

→ $1{,}xxxxxxxxx \cdot 2^{yyyy}$ bzw. $0{,}xxxxxxxxx \cdot 2^{yyyy}$

Eine Gleitkommazahl wird dann durch eine Bitsequenz fester Länge (üblicherweise mit einem Vielfachen der Wortlänge des Computers) dargestellt.

Der Exponent ist häufig auch in 2er-Komplementdarstellung.

Beispiel: 32 Bit-Darstellung



Die reelle Zahl bestimmt sich bei einer führenden **Null** aus:

$$(-1)^S \times Mantisse \times 2^{Exponent}$$

→ Wertebereich (bei vorzeichenbehaftetem Exponenten):

$$-2 \cdot 10^{38} \ldots 2 \cdot 10^{38}$$

Kleinste darstellbare Zahl:

$$2 \cdot 10^{-38}$$

Die reelle Zahl bestimmt sich bei einer führenden **Eins** aus:

$$(-1)^S \times (1 + Mantisse) \times 2^{Exponent}$$

Zahlenbeispiel:

In Dezimaldarstellung:

$$-0{,}75 = -3/4 = -3/2^2$$

In Binärdarstellung:

$$-11/2^2 = -0{,}11 = -0{,}11 \cdot 2^0 = -1{,}1 \cdot 2^{-1}$$

→ Darstellung bei führender Null:

1 **|** 000.0000.0 **|** 110.0000.0000.0000.0000.0000

Darstellung bei führender Eins:

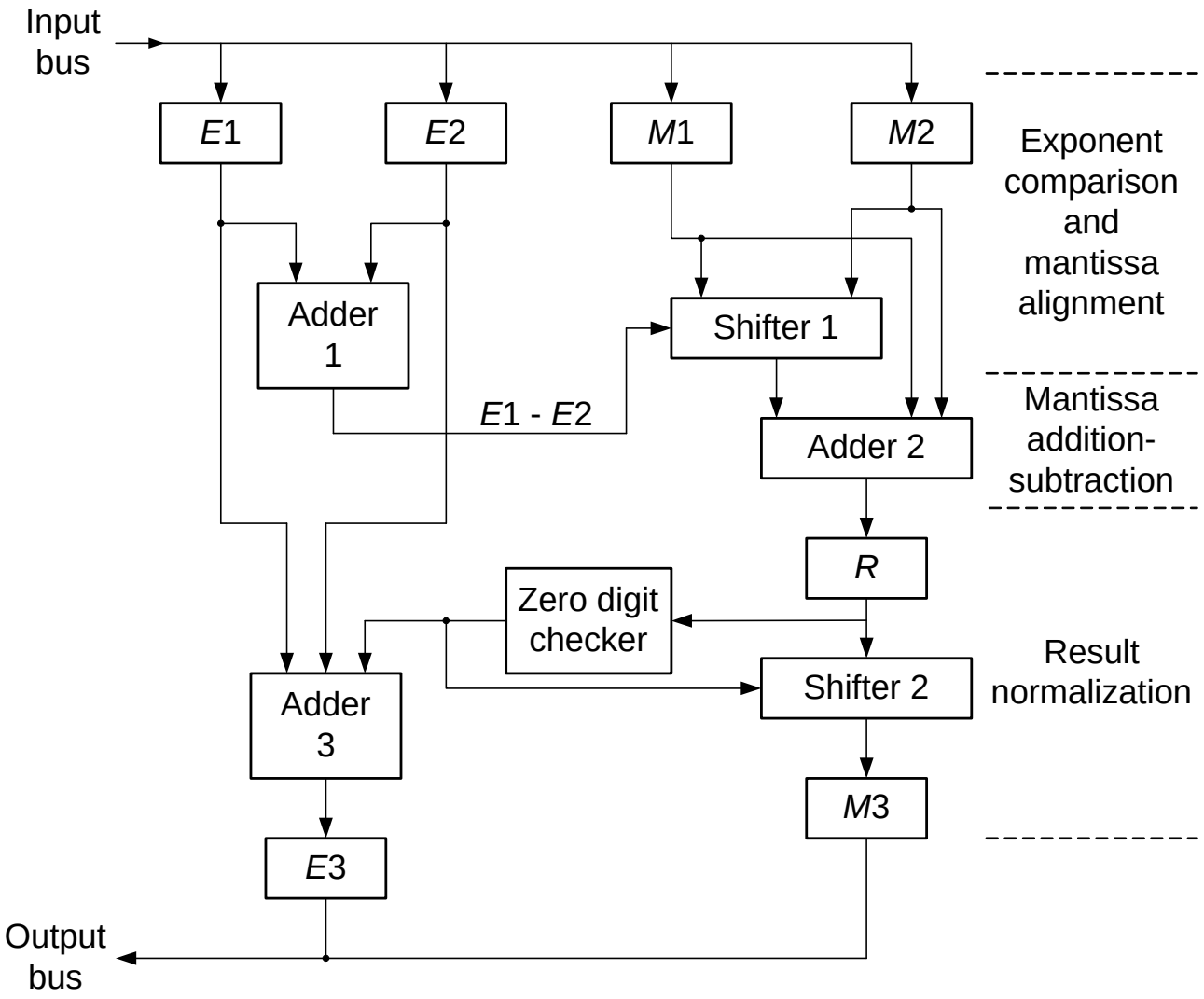
1 **|** 111.1111.1 **|** 100.0000.0000.0000.0000.0000

Rechnen mit Gleitkommazahlen

Zum Addieren bzw. Subtrahieren von Gleitkommazahlen muss zunächst der Exponent durch Schieben einer der Mantissen (Normalisieren) angeglichen werden.

Ggf. ist ein weiterer Normalisierungsschritt nach der Berechnung nötig, um das Komma wieder an die richtige Stelle zu rücken.

Beispiel: Blockschaltbild eines Gleitkomma-Rechenwerks



Getrenntes Exponenten- und Mantissenrechenwerk.

Ein Steuerwerk steuert die einzelnen Schritte der Gleitkommaoperationen.

z. B. bei einer Floating Point-Addition:

- Exponentenvergleich,
- Exponentenangleich durch Rechts-/Linksshift der Mantisse,
- Addition der Mantissen,
- Normalisierung

2.7 Codes

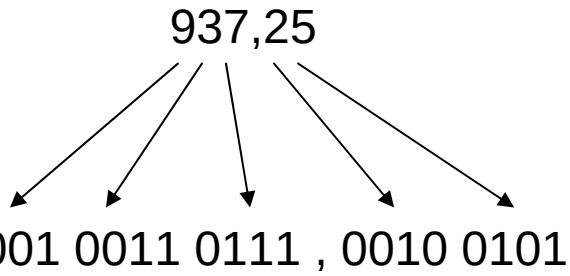
Obwohl die meisten Computer mit Dualzahlen arbeiten, werden an den Schnittstellen oft andere Formate (Codes) verwendet, die anwendungsabhängig vorgegeben werden.

DIN 44300: Ein Code ist eine Vorschrift für die Zuordnung der Zeichen eines Zeichenvorrats zu denjenigen eines anderen Zeichenvorrats.

So kann die Umwandlung der Darstellung von Dezimalzahlen in Dual- oder Hexadezimalzahlen und umgekehrt als Umcodierung aufgefasst werden.

Sehr häufig wird die Dezimaldarstellung verwendet. Dabei wird jede Stelle / Ziffer der Dezimalzahl durch eine Dualzahl repräsentiert. Dann spricht man von der BCD-Codierung (*binary coded decimal*; binär-dezimaler Code).

Beispiel:



Der BCD-Code ist ein gewichteter Code. D.h., die Dezimalziffer *N* ergibt sich durch eine gewichtete Addition der einzelnen Bitstellen a_i.

$$N = w_3 \cdot a_3 + w_2 \cdot a_2 + w_1 \cdot a_1 + w_0 \cdot a_0$$

Es sind auch andere Codierungen (mit anderen Gewichten) gebräuchlich.

Dezimal-ziffer	8-4-2-1-Code (**BCD**)	6-3-1-1-Code	Excess-3-Code	(2-aus-5)-Code	Gray Code
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

Der 6-3-1-1-Code wird wie der BCD-Code gebildet, allerdings mit den Gewichten 6,3,1,1 anstatt 8,4,2,1.

Der Excess-3-Code (´Stibitz-Code´) ergibt sich aus dem BCD-Code, indem eine ´3´ zu jedem Codewort addiert wird. Er ist negationssymmetrisch und dadurch rechenfähig. Außerdem werden die Codeworte ´0000´ und ´1111´ vermieden. Das unterstützt das Erkennen von Hardwarefehlern, die häufig zu einem dieser Codeworte führen.

Die 2-aus-5-Codierung ist eine Variante der (m-aus-n)-Codes, bei der immer 2 der 5 Bits auf ´1´ gesetzt sind. Dadurch hat dieser Code gute Eigenschaften zur Fehlerprüfung:

- Einfachfehler werden immer erkannt.
- Doppelfehler werden nur erkannt, wenn sie einseitig sind.
- Dreifachfehler werden als solche erkannt, wenn nur Verfälschungen von ´0´ in ´1´ vorliegen. Sonst werden sie als Einfachfehler gewertet.

Der Gray-Code ist ein einschrittiger Code. Bei ihm unterscheiden sich aufeinander folgende Codeworte nur in einer Bitstelle. Sie werden daher gerne in Zählschaltungen eingesetzt (Abtastsicherheit beim Übergang von einem Zeichen zum nächsten).

Nicht gewichtete Codes wie der 2-aus-5-Code oder der Gray-Code lassen sich nicht durch eine einfache Berechnung aus dem Dezimalwert ableiten.

Weitere Codes

Bei der Darstellung, Übertragung und Speicherung von Informationen werden neben Zahlen auch andere Zeichen benötigt, z.B. bei Text die des Alphabets (alphanumerischen Zeichen) oder Satz- und Steuerzeichen. Ihre Codierung benötigt mehr als vier Bits.

ASCII-Code

Als internationaler Standard für die alphanumerische Codierung wird üblicherweise der ASCII-Code (American Standard Code for Information Interchange) verwendet.

	$B_7B_6B_5$							
$B_4B_3B_2B_1$	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control Characters:

NULL	NULL	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete