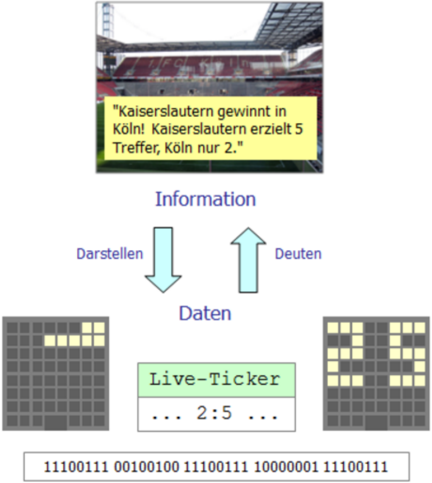
Codierung ganzer Zahlen

Informatik

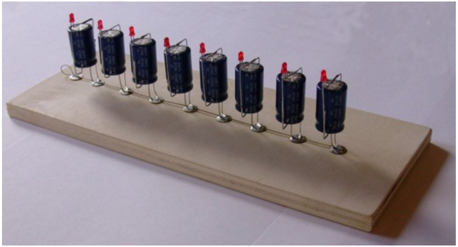
Information und Daten



Information muss immer in geeigneter Weise dargestellt werden, um sie als **Daten** maschinell weiterverarbeiten zu können.
Aus Daten gewinnt man erst dann Information, wenn sie gedeutet werden können.

Zur Darstellung von Information nutzt man häufig Systeme, die nur zwei Zustände einnehmen können: an/aus; geladen/ungeladen; Strom fließt/Strom fließt nicht; magnetisiert/unmagnetisiert.

Ein einfacher Speicher, bei dem Information mit Hilfe (un)geladener Kondensatoren dargestellt wird:



Die beiden Zustände eines Zweizustandssystems werden in der Regel mit Hilfe der beiden Ziffern 0 und 1 beschrieben.

Unter einem **Bit** versteht man eine Einheit zur Informationsdarstellung, die nur zwei Werte annehmen kann: 0 und 1.

Unter einem **Byte** versteht man eine Einheit aus 8 Bits.

1 Byte	8 Bit
1 KiloByte (KB)	1000 Byte
1 Megabyte (MB)	1000 KB
1 Gigabyte (GB)	1000 MB

Unter einem **Bit** versteht man eine Einheit zur Informationsdarstellung, die nur zwei Werte annehmen kann: 0 und 1.

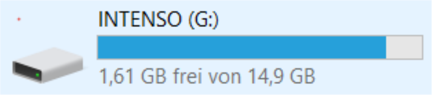
Unter einem **Byte** versteht man eine Einheit aus 8 Bits.

1 Byte	8 Bit
1 KiloByte (KB)	1000 Byte
1 Megabyte (MB)	1000 KB
1 Gigabyte (GB)	1000 MB

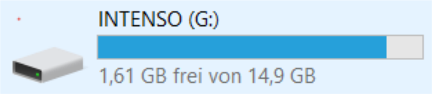
Diese Einheiten bauen auf Zweierpotenzen statt Zehnerpotenzen auf:

1 KibiByte (KiB)	1024 Byte
1 Mebibyte (MiB)	1024 KiB
1 Gibibyte (GiB)	1024 MiB

Ein 16GB USB-Stick im Windows-Explorer:



Ein 16GB USB-Stick im Windows-Explorer:




```
# konvertiert gigabyte in gibibyte
def gb2gib(x):
    bytes = 1000 * 1000 * 1000 * x
    return round(bytes / (1024 * 1024 * 1024),2)

>>> gb2gib(16)
14.9
```

Werden die Daten nur mit Bits dargestellt spricht man von einer **Binärdarstellung der Daten**.

Im folgenden geht es um die Binärdarstellung von ganzen Zahlen.

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

$(4719)_{10} =$

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

$(4719)_{10} = 9 \cdot 10^0 + 1 \cdot 10^1 + 7 \cdot 10^2 + 4 \cdot 10^3$
$(273)_8 =$

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

$$(4719)_{10} = 9 \cdot 10^0 + 1 \cdot 10^1 + 7 \cdot 10^2 + 4 \cdot 10^3$$
$$(273)_8 = 3 \cdot 8^0 + 7 \cdot 8^1 + 2 \cdot 8^2 = (187)_{10}$$
$$(10010)_2 =$$

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

$(4719)_{10} = 9 \cdot 10^0 + 1 \cdot 10^1 + 7 \cdot 10^2 + 4 \cdot 10^3$
$(273)_8 = 3 \cdot 8^0 + 7 \cdot 8^1 + 2 \cdot 8^2 = (187)_{10}$
$(10010)_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = (18)_{10}$
$(E52F)_{16} =$

Dezimalzahlen	10 Ziffern: $0, 1, 2, ...9$	4719
Dualzahlen	2 Ziffern: $0, 1$	10010
Oktalzahlen	8 Ziffern: $0, 1, 2, ...7$	273
Hexadezimalzahlen	16 Ziffern: $0, 1, 2, ...9, A, B, C, D, E, F$	$E52F$

$$(4719)_{10} = 9 \cdot 10^0 + 1 \cdot 10^1 + 7 \cdot 10^2 + 4 \cdot 10^3$$
$$(273)_8 = 3 \cdot 8^0 + 7 \cdot 8^1 + 2 \cdot 8^2 = (187)_{10}$$
$$(10010)_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = (18)_{10}$$
$$(E52F)_{16} = 15 \cdot 16^0 + 2 \cdot 16^1 + 5 \cdot 16^2 + 14 \cdot 16^3 = (58671)_{10}$$

Umwandlung Dualzahl in Dezimalzahl

```
def dual_dez(s):
    '''
    s: String aus 0 und 1
    returns: Dezimalzahl, die der Dualzahl s entspicht
    '''
```

Umwandlung Dualzahl in Dezimalzahl

```
def dual_dez(s):
    '''
    s: String aus 0 und 1
    returns: Dezimalzahl, die der Dualzahl s entspicht
    '''
    x = 0
    sw = 1    # Stellenwert
    for c in s[::-1]:
        if c == '1':
            x+=sw
        sw*=2
    return x
```

Aufruf:

```
>>> dual_dez('1101')
13
```

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2:

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 =$$
$$(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2:

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2: rechts kommt noch eine 0 dran.

10110 22
1011

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 =$$
$$(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2: rechts kommt noch eine 0 dran.

10110	22
1011	11
101	

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2: rechts kommt noch eine 0 dran.

10110	22
1011	11
101	5
10	

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2: rechts kommt noch eine 0 dran.

10110	22
1011	11
101	5
10	2
101100	

$$10110/2 = (1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)/2 = (1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = 1011$$

ganzzahlige Division durch 2: die rechte Ziffer verschwindet
Multiplikation mit 2: rechts kommt noch eine 0 dran.

10110	22
1011	11
101	5
10	2
101100	44

Umrechnung von 13 in eine Dualzahl

xxxx 13

Umrechnung von 13 in eine Dualzahl

xxxx	13	1
xxx	6	0
xx	3	1
x	1	1
	0	

Dualzahl ergibt sich von unten nach oben: 1101

Unsere Schreibweise zur Umrechnung:

13	
6	1
3	0
1	1
0	1

Dezimalzahl 13 ist Dualzahl 1101

41

Unsere Schreibweise zur Umrechnung:

13	
6	1
3	0
1	1
0	1

Dezimalzahl 13 ist Dualzahl 1101

41	
20	1
10	0
5	0
2	1
1	0
0	1

Dezimalzahl 41 ist Dualzahl 101001

Umrechnung einer Dezimalzahl in eine Oktalzahl

41

Umrechnung einer Dezimalzahl in eine Oktalzahl

41	
5	1
0	5

Dezimalzahl 41 ist Oktalzahl 51

Umrechnung einer Dezimalzahl in eine Oktalzahl

41	
5	1
0	5

Dezimalzahl 41 ist Oktalzahl 51

Umrechnung einer Dezimalzahl in eine Hexadezimalzahl

3882

Umrechnung einer Dezimalzahl in eine Oktalzahl

41	
5	1
0	5

Dezimalzahl 41 ist Oktalzahl 51

Umrechnung einer Dezimalzahl in eine Hexadezimalzahl

3882	
242	A
15	2
0	F

Dezimalzahl 3882 ist Hexadezimalzahl F2A

Umrechnung Dezimalzahl in Dualzahl

```
def dez_dual(x):
```

Umrechnung Dezimalzahl in Dualzahl

```
def dez_dual(x):
    if x == 0: return '0'
    s = ""
    while x != 0:
        s = str(x%2) + s
        x = x // 2
    return s
```

Aufruf:

```
>>> dez_dual(47)
'101111'
```

Die Addition von Dualzahlen erfolgt analog zum Dezimalsystem. Die Stellenwerte werden addiert, gegebenenfalls mit Übertrag.

$$\begin{array}{cccccc} & 0 & 1 & 0 & 1 & 0 \\ + & 1 & 1 & 0 & 1 & 1 \end{array}$$

Die Addition von Dualzahlen erfolgt analog zum Dezimalsystem. Die Stellenwerte werden addiert, gegebenenfalls mit Übertrag.

$$
\begin{array}{cccccc}
 & 0 & 1 & 0 & 1 & 0 \\
+ & 1 & 1 & 0 & 1 & 1 \\
 & & & & & \\
1 & 0 & 0 & 1 & 0 & 1
\end{array}
$$

Mit 3 bit können die Zahlen von 0-7 dargestellt werden

1	1	1	7
1	1	0	6
1	0	1	5
1	0	0	4
0	1	1	3
0	1	0	2
0	0	1	1
0	0	0	0

Wie kann man negative Zahlen darstellen ?

0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0

1	1	1	1	-7
1	1	1	0	-6
1	1	0	1	-5
1	1	0	0	-4
1	0	1	1	-3
1	0	1	0	-2
1	0	0	1	-1
1	0	0	0	-0

SO NICHT

0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

Die 4-bit Zweierkomplement Darstellung von $-8 = -2^3$ bis $7 = 2^3 - 1$.

Wertebereiche:

	Dualzahl	Zweierkomplement
4-Bit		
8-Bit		

Einige Binärdarstellungen im Zweierkomplement:

	4-Bit	8-Bit
größte Zahl		
kleinste Zahl		
1		
-1		

Wertebereiche:

	Dualzahl	Zweierkomplement
4-Bit	0 ... 15	-8 ... +7
8-Bit	0 ... 255	-128 ... +127

Einige Binärdarstellungen im Zweierkomplement:

	4-Bit	8-Bit
größte Zahl	0111	0111 1111
kleinste Zahl	1000	1000 0000
1	0001	0000 0001
-1	1111	1111 1111

Im Zweierkomplement kann für Addition und Subtraktion derselbe Algorithmus verwendet werden.

Codierung von -x:

```
codiere x
negiere bitweise
addiere 1
```

Im Zweierkomplement kann für Addition und Subtraktion derselbe Algorithmus verwendet werden.

Codierung von -x:

```
codiere x
negiere bitweise
addiere 1
```

Codierung von -5 (4 bit)

Im Zweierkomplement kann für Addition und Subtraktion derselbe Algorithmus verwendet werden.

Codierung von -x:

```
codiere x
negiere bitweise
addiere 1
```

Codierung von -5 (4 bit)

5	0101
bitweise Negation	1010
addiere 1	0001
-5	1011

Im Zweierkomplement kann für Addition und Subtraktion derselbe Algorithmus verwendet werden.

Codierung von -x:

```
codiere x
negiere bitweise
addiere 1
```

Codierung von -5 (4 bit)

5	0101
bitweise Negation	1010
addiere 1	0001
-5	1011

Codierung von -100 (8 bit)

Im Zweierkomplement kann für Addition und Subtraktion derselbe Algorithmus verwendet werden.

Codierung von -x:

```
codiere x
negiere bitweise
addiere 1
```

Codierung von -5 (4 bit)

5	0101
bitweise Negation	1010
addiere 1	0001
-5	1011

Codierung von -100 (8 bit)

100	01100100
bitweise Negation	10011011
addiere 1	00000001
-100	10011100

Wenn die Verknüpfung die zulässigen Wertebereiche verlässt, entstehen falsche Ergebnisse.

In der Programmiersprache Java werden ganze Zahlen vom Typ `int` intern im 32-bit Zweierkomplement gespeichert.

Was macht dieses Programm?

```
int k = 1;
while (k > 0)
  k = k+1;
System.out.println(k);
```

In der Programmiersprache Java werden ganze Zahlen vom Typ `int` intern im 32-bit Zweierkomplement gespeichert.

Was macht dieses Programm?

```
int k = 1;
while (k > 0)
  k = k+1;
System.out.println(k);
```

-2147483648

In der Programmiersprache Java werden ganze Zahlen vom Typ `int` intern im 32-bit Zweierkomplement gespeichert.

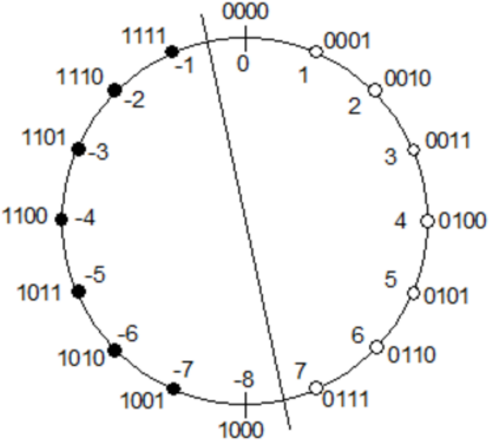
Was macht dieses Programm?

```
int k = 1;
while (k > 0)
  k = k+1;
System.out.println(k);
```

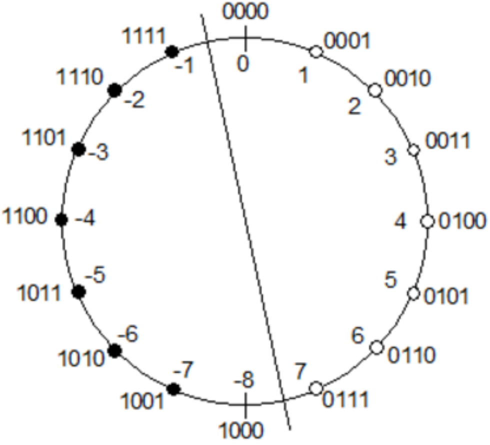
```
-2147483648
```

$-2147483648 = -2^{31}$

Darstellung des Zweierkomplements im Zahlenkreis:



Darstellung des Zweierkomplements im Zahlenkreis:




```
>>> bin(16-5)
'0b1011'
>>> bin(256-100)
'0b10011100'
>>>
```