

# Kapitel 2

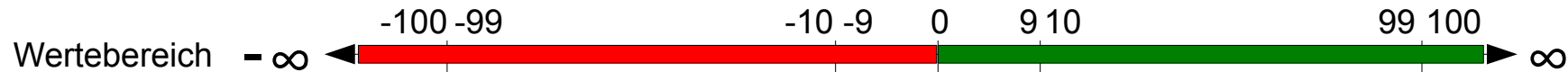
## Zahlensysteme

# Zahlensysteme

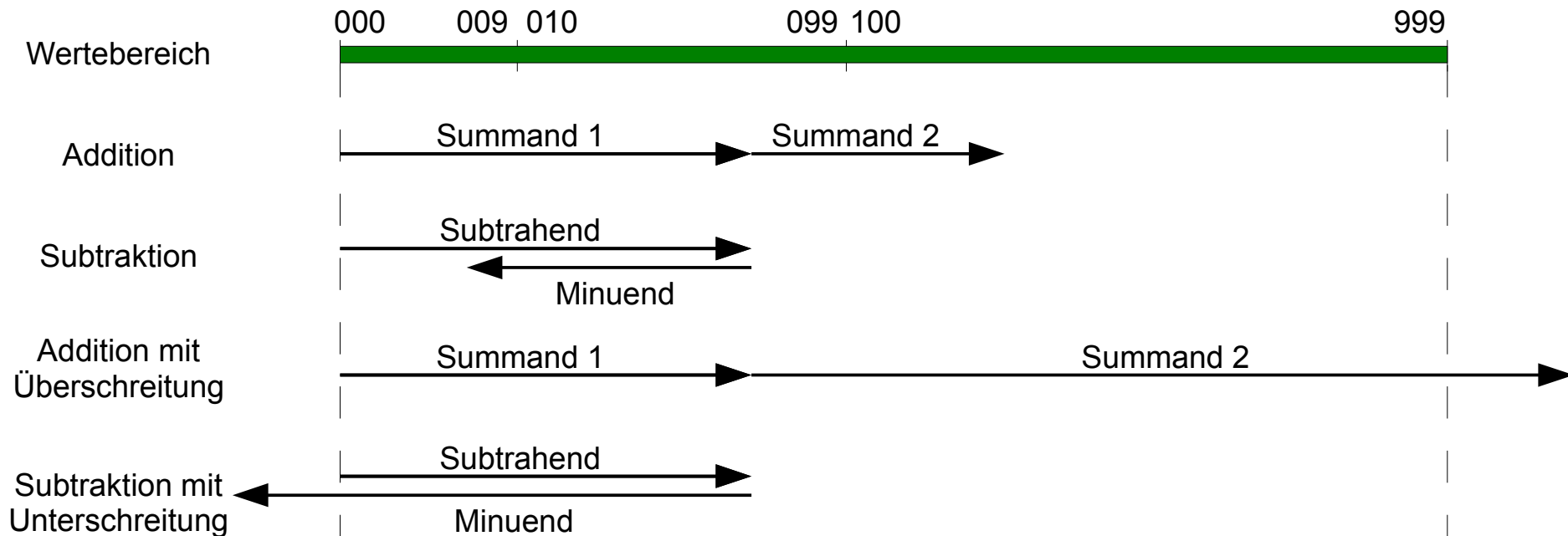
- Definitionen
  - **Ziffern** : Zeichen zur Darstellung von Zahlen
  - **Zahl** : Eine Folge von Ziffern
  - **Basis eines Zahlensystems** : Anzahl der unterschiedlichen Ziffern
- Moderne Zahlensysteme sind **positionale** Systeme
  - Die Position einer Ziffer innerhalb einer Zahl bestimmt den Beitrag, den diese Ziffer zum Wert der Zahl liefert.
- Dezimalsystem → Basis : **10**
  - Ziffern : **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
  - z.B.: dezimal  $123 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0$
- Binärsystem → Basis : **2**
  - Ziffern : **0, 1** (binary digit → Bit)
  - z.B.: binär  $'1001' = 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$
- Hexadezimalsystem → Basis : **16**
  - Ziffern : **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**
  - z.B.: Hex-Zahl  $'1AF' = 1 * 16^2 + 10 * 16^1 + 15 * 16^0$
  - Verwandtschaft : 4 Bits ↔ 1 Hexziffer (Nibble)
  - (z.B.: binär  $'0011 1011' = \text{hexadezimal } '3B'$ )

# Dezimalsystem

## Unser Dezimalsystem

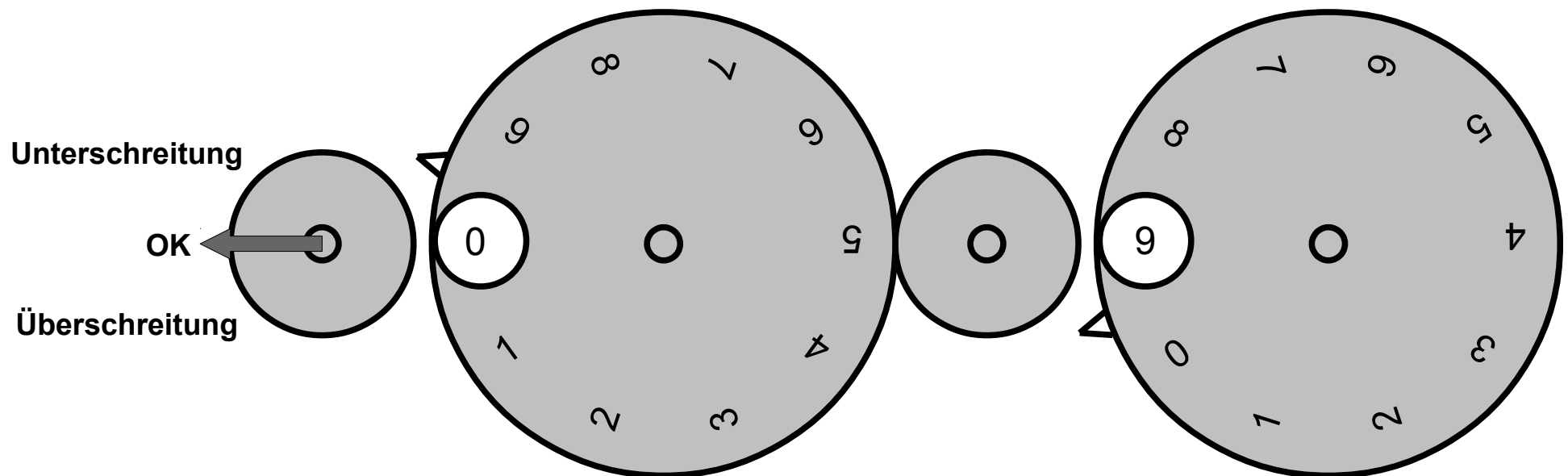


## Leibniz-Rechner (nur positive Werte, z.B. 3-stellig)



- Die Beschränkung des Wertebereichs hat zur Folge, dass überprüft werden muss, ob bei einer Addition/Subtraktion der gültige Wertebereich verlassen wird.

# Realisierung in einem mechanischen Rechner



- **Addition** : Drehen der Räder im Uhrzeigersinn
  - Übergang von 9 nach 0 erzeugt einen Übertrag (**Carry**)
- **Subtraktion** : Drehen der Räder entgegen dem Uhrzeigersinn
  - Übergang von 0 nach 9 erzeugt einen **Borrow**
- Ein Carry/Borrow aus einer Position muss an die nächst höhere Position weitergegeben werden, solange es noch eine höhere Position gibt.
  - z.B. Addition von 1 am rechten Rad der Abbildung.
- Ein Carry/Borrow aus der höchsten Position signalisiert eine Überschreitung bzw. Unterschreitung des Wertebereichs.
  - z.B. Subtraktion von 1 am linken Rad der Abbildung.

# Negative Dezimalzahlen

## Negative Zahlen in unserem Dezimalsystem

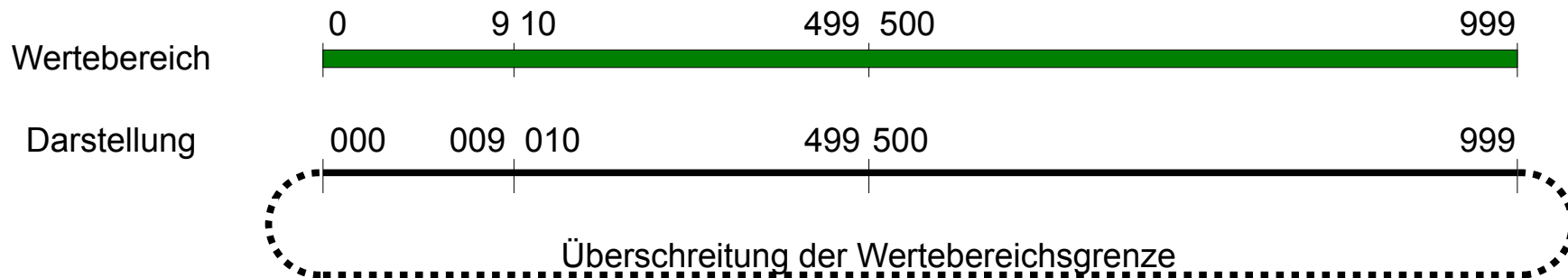
- Werden durch das **Vorzeichen** spezifiziert
- Erfordern ein 'kompliziertes' Subtraktionsverfahren

$$\begin{array}{r}
 \text{Minuend >} \\
 \text{Subtrahend !} \\
 142 \\
 - 136 \\
 \hline
 -006
 \end{array}$$

$$\begin{array}{r}
 136 \\
 - 142 \\
 \hline
 \text{Minuend <} \\
 \text{Subtrahend !} \\
 -006
 \end{array}$$

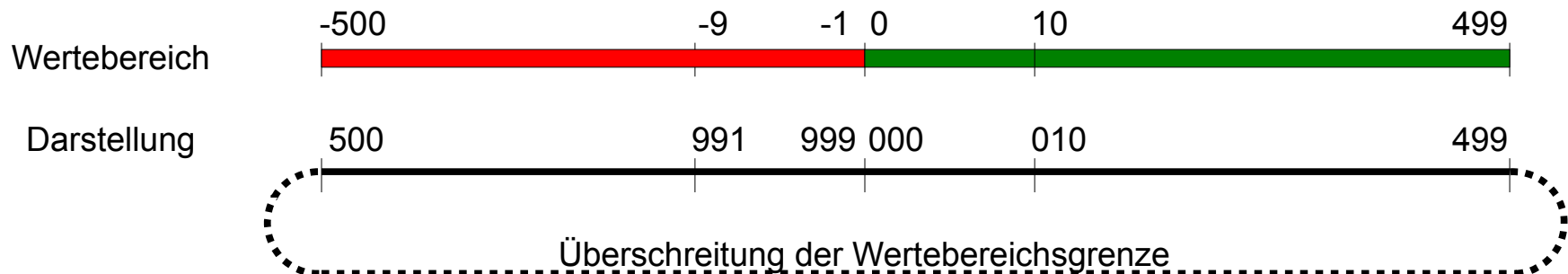
- Realisierung würde viele Schaltkreise erfordern

# Negative Zahlen beim Leibniz-Rechner (nur theoretisch)



- Von den 1000 Werten des 3-stelligen Leibniz-Rechners werden 500 für negative Zahlen reserviert
- Die Darstellungen für die positiven Zahlen und die Zahl '0' sollten die selben sein wie bei den rein positiven Zahlen.
- Für Addition und Subtraktion sollte der selbe Mechanismus verwendet werden wie bei den rein positiven Zahlen.

0 0 0	9 9 9	9 9 8	5 0 0
- 1	- 1	- 4 9 8	- 1
9 9 9	9 9 8	5 0 0	4 9 9



- Die erste Ziffer einer Zahl ist 0 ... 4 : positive Zahl (inkl. 0); die erste Ziffer ist 5 ... 9 : negative Zahl

## Negative Zahlen beim Leibniz-Rechner (Forts.)

- Der Übergang von '999' nach '000' bei einer Addition und der Übergang von '000' nach '999' bei einer Subtraktion stellt **kein** Verlassen des Wertebereichs dar.
  - Ein Carry oder Borrow signalisiert lediglich den Übergang von positiven zu negativen Zahlen und umgekehrt
- Der Übergang von '499' nach '500' bei einer Addition und der Übergang von '500' nach '499' bei einer Subtraktion stellt dagegen ein Verlassen des Wertebereichs dar.
  - Diese Übergänge werden als **Overflow** bezeichnet.
- Um sicher zu stellen, dass der gültige Wertebereich nicht verlassen wurde, überprüft man beim Rechnen mit
  - **rein positiven** Zahlen auf **Carry/Borrow**,
  - **gemischt positiven und negativen** Zahlen auf **Overflow**.

### Berechnung des Betrags einer negativen Zahl N

- $|N| = 1000 - N = 999 - N + 1 = (\text{Neunerkomplement von } N) + 1$
- z.B.:  $|999| = 1000 - 999 = 1; \quad (000) + 1 = 1$

# Binär- und Hexadezimalsystem

- Informationsgehalt
  - 1 Bit :  $2^1 = 2$  Werte → JA – NEIN; Ein - Aus
  - 4 Bit :  $2^4 = 16$  Werte → Dezimalziffern (BCD Code)
  - 6 Bit :  $2^6 = 64$  Werte → Ziffern, Großbuchstaben, Sonderzeichen
  - 8 Bit :  $2^8 = 256$  Werte → alle Zeichen, auch nationale Sonderzeichen (ASCII, EBCDIC)



# BYTE

- 1 Byte = 8 Bits = 2 Hexziffern
- $2^8 = 256$  unterschiedliche Werte darstellbar
- Kleinste im Speicher adressierbare Informationseinheit (in der Regel)
- Verwendung
  - Zahlenwerte (z.B.: C unsigned char)
  - Zeichen (z.B.: ASCII-Code : 0x30 = '0', 0x31 = '1', 0x39 = '9', 0x41 = 'A', 0x61 = 'a' (siehe Tabelle im Script S.116))
  - Binärwerte (z.B.: Maschinencode)
- Bit Bezeichnungen (8051 Konvention)

0000 0000B ... 1111 1111B = 0x00 ... 0xFF  
 ↑            ↑        ↑        ↑  
 Bit 7      Bit 0     $2^7$  .....  $2^0$

- Bit 7 : MSB = Most Significant Bit
- Bit 0 : LSB = Least Significant Bit

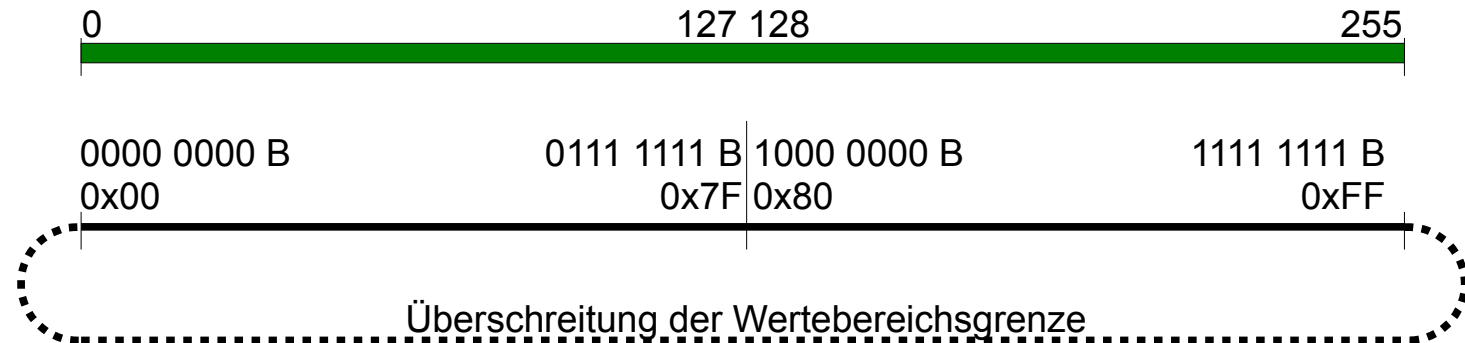
## Schreibweise für Hexadezimalzahlen

z.B.: Intel : 5AH, 0F6H;    Motorola/Keil : 0x5A, 0xF6;    IBM : X'5A', X'F6';    Borland : \$5A, \$0F6

# Binärzahlen

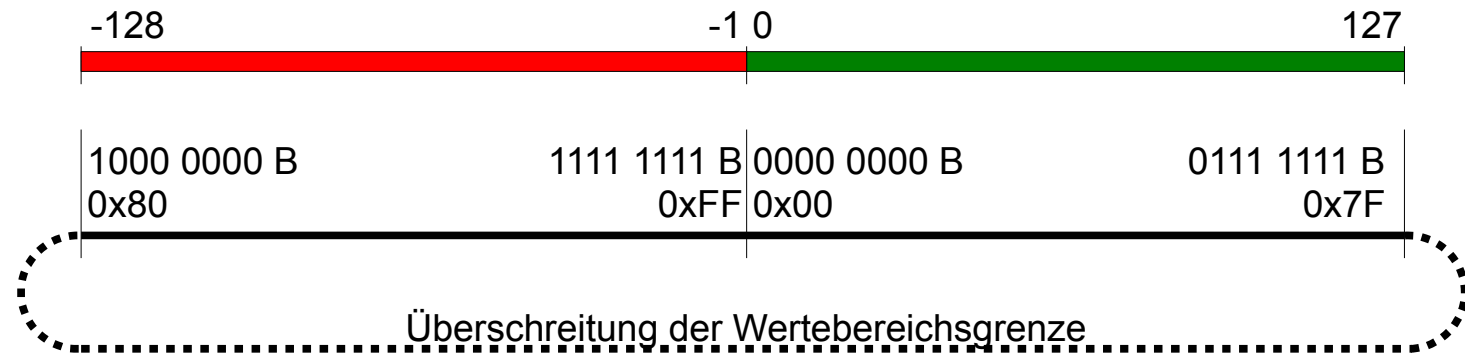
## Binärzahlen ohne Vorzeichen - 1 Byte

(Unsigned Binary – unsigned char)



## Binärzahlen mit Vorzeichen - 1 Byte

(Signed Binary – signed char)



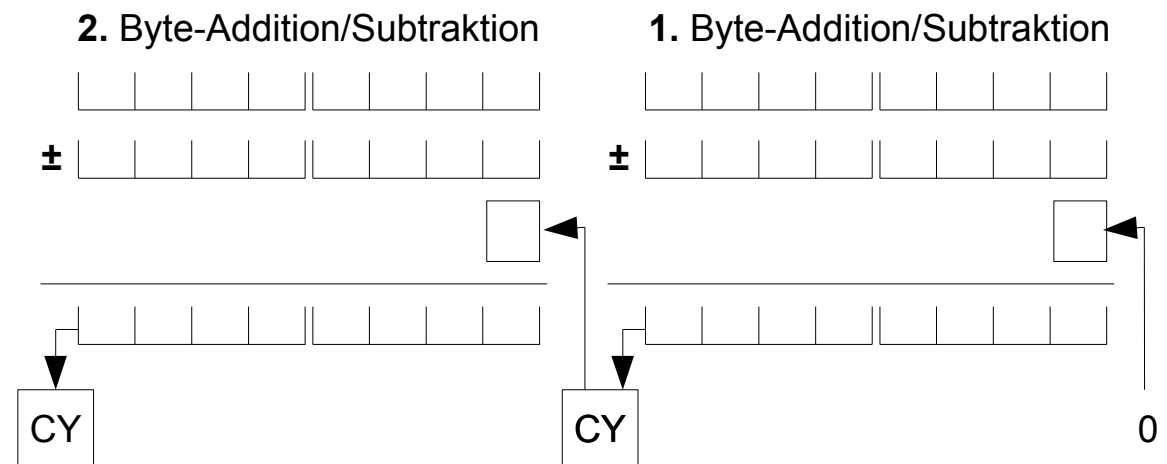
Erstes Bit einer Binärzahl = 0 → positive Zahl; erstes Bit einer Binärzahl = 1 → Negative Zahl

## Binärzahlen (Forts.)

- Wie bei den Dezimalzahlen ergibt sich auch bei Binärzahlen ein Überschreiten des Wertebereichs, wenn
  - bei unsigned binary Zahlen ein Übergang zwischen 255 und 0 (0xFF ↔ 0x00) stattfindet,
    - und zwar ein Carry bzw. Borrow
  - bei signed binary Zahlen ein Übergang zwischen 127 und 128 (0x7F ↔ 0x80) stattfindet.
    - und zwar ein Overflow
- Der Prozessor registriert in einem einzigen Indikator, dem **Carry Flag (CY)**, ob ein Carry oder Borrow aufgetreten ist.
  - Der Programmierer kann Carry und Borrow unterscheiden, denn er weiß, ob zuvor eine Addition oder eine Subtraktion durchgeführt wurde.
- Dazu gibt es noch einen Indikator für Overflows, das **Overflow Flag (OV)**.
- Ein Prozessor führt Additionen und Subtraktionen für Unsigned Zahlen und Signed Zahlen in exakt gleicher Weise aus.
  - Er setzt ggf. das Carry Flag, als ob er Unsigned Zahlen verarbeiten würde.
  - Er setzt ggf. das Overflow Flag, als ob er Signed Zahlen verarbeiten würde
  - Da der Programmierer weiß, welcher Datentyp verarbeitet wurde, kann er durch Inspektion des Carry Flags und/oder des Overflow Flags überprüfen, ob der gültige Wertebereich überschritten wurde.

## Binärzahlen (Forts.)

- Bei der Addition/Subtraktion von Zahlen, die aus **mehreren** Bytes bestehen, werden die Bytes einzeln von rechts nach links verarbeitet (von niederwertig nach höherwertig).
  - Bei jeder Byte Operation wird der sog. **Carry Out** Wert im Carry Flag gespeichert.
  - Am Beginn einer Operation wird der Wert des Carry Flags als **Carry In** in die Berechnung einbezogen.
  - Der Carry Out aus einer Byte Position wird so als Carry In in die nächst höhere Byte-Position übernommen.
  - Der Carry In für das niederwertigste Byte wird auf 0 forciert.
  - Nur die Werte des Carry Flags und des Overflow Flags aus der **höchsten** Position sind für die Gesamtoperation relevant.
- Bei der Addition/Subtraktion **jedes** einzelnen Bytes setzt der Prozessor sowohl das Carry Flag als auch das Overflow Flag auf 0 bzw 1.
- Ob ein gesetztes Carry Flag **oder** Overflow Flag aus der höchsten Position relevant ist, hängt davon ab, ob die Operation für Unsigned Zahlen oder **oder** Signed Zahlen durchgeführt wurde.



## Overflow

- Ein Overflow tritt immer dann auf, wenn sich bei der Addition/Subtraktion der höchstwertigen Bytes ein Carry aus Bitposition 6 oder Bitposition 7, aber nicht aus beiden ergibt.

Carry aus Bitposition 7	Carry aus Bitposition 6	Overflow
0	0	0
0	1	1
1	0	1
1	1	0

- Bei einer vorzeichenbehafteten Binärzahl stellt das höchstwertige Bit das Vorzeichen dar.
- Die anderen Bits bestimmen den Wert der Zahl.
- Der Wert des Carry Indikators aus der Bitposition 6 des höchstwertigen Bytes unterscheidet, ob der gültige Wertebereich verlassen wurde.
  - Bei der Addition zweier positiver Zahlen bedeutet **ein Carry** aus Bitposition 6 einen Overflow, denn
    - Bit 7 der beiden Summanden = 0 → Summe dieser beiden Bits ergibt nie einen Carry.
    - Addition zweier negativer Zahlen entsprechend.
- Addition einer positiven und einer negativen Zahl erzeugt nie einen Overflow.
- Die nachfolgende Tabelle zeigt einige charakteristische Beispiele.

## Binärzahlen (Forts.)

Operation	Ergebnis	Carry <sub>Bit7</sub>	Carry <sub>Bit6</sub>	Carry	Unsigned Zahlen	Overflow	Signed Zahlen
0x00 + 0x7F	0x7F	0	0	0	0 + 127 = 127	0	0 + 127 = 127
0x00 + 0x80	0x80	0	0	0	0 + 128 = 128	0	0 + (-128) = -128
0x00 + 0xFF	0xFF	0	0	0	0 + 255 = 255	0	0 + (-1) = -1
0x7F + 0x01	0x80	0	1	0	127 + 1 = 128	1	127 + 1 = [-128]
0x7F + 0x80	0xFF	0	0	0	127 + 128 = 255	0	127 + (-128) = -1
0x7F + 0x81	0x00	1	1	1	127 + 129 = [256]	0	127 + (-127) = 0
0x7F + 0xFF	0x7E	1	1	1	127 + 255 = [126]	0	127 + (-1) = 126
0xFF + 0xFF	0xFE	1	1	1	127 + 127 = [254]	0	(-1) + (-1) = -2
0x00 - 0x01	0xFF	1	1	0	0 - 1 = [-1]	0	0 - 1 = -1
0x80 - 0x01	0x7F	0	1	0	128 - 1 = 127	1	(-128) - 1 = [127]

- Bei Erweiterung des Wertebereiches (z.B. Casting von *char* nach *int*) werden
  - positive Zahlen links mit '0' erweitert,
  - negative Zahlen links mit '1' erweitert (z.B.: -128 = 1111 1111 1000 0000B)
- Umrechnung einer positiven Zahl in eine negative und umgekehrt :
  - $|N| = 1\ 0000\ 0000 - N = 0\ 1111\ 1111 - N + 1 = (\text{Einerkomplement von } N) + 1$
  - z.B.:  $|1111\ 1110| = 1\ 0000\ 0000 - 0\ 1111\ 1110 + 1 = 0000\ 0001 + 0000\ 0001 = 0000\ 0010$ 

```

1 0000 0000
- 0 1111 1110
-----
0 0000 0010
          
```

# Speicherformate

Für Informationseinheiten, die aus zwei Bytes (8051, ...) 4 By, 8 By (andere Architekturen) bestehen, gibt es zwei Formate für die Anordnung dieser Gruppe von Bytes im Speicher :

Beim 8051 Prozessor besteht ein Integerwert aus 16 Bits :

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
-----	-----	-----	-----	-----	-----	----	----	----	----	----	----	----	----	----	----

## ● Big Endian :

○ 'Das große Ende zuerst' :

B15	B14	B13	B12	B11	B10	B9	B8		B7	B6	B5	B4	B3	B2	B1	B0
-----	-----	-----	-----	-----	-----	----	----	--	----	----	----	----	----	----	----	----

○ Beispiel: Der Integerwert **0x1AB4** in Adresse 61 und 62

Speicheradresse      5F      60      61      62      63      64

		<b>1A</b>	<b>B4</b>		
--	--	-----------	-----------	--	--

○ Verwendung : Motorola 68000, PowerPC (zum Teil umschaltbar), KEIL C Compiler

## ● Little Endian :

○ 'Das kleine Ende zuerst' :

B7	B6	B5	B4	B3	B2	B1	B0		B15	B14	B13	B12	B11	B10	B9	B8
----	----	----	----	----	----	----	----	--	-----	-----	-----	-----	-----	-----	----	----

○ Beispiel: Der Integerwert **0x1AB4** in Adresse 61 und 62

Speicheradresse      5F      60      61      62      63      64

		<b>B4</b>	<b>1A</b>		
--	--	-----------	-----------	--	--

○ Verwendung : Intel, AMD (umschaltbar)

# Wichtige Binärzahlen

<b>dezimal</b>	0	1	2	3	4	5	6	7
<b>binär</b>	0000	0001	0010	0011	0100	0101	0110	0111
<b>hexadezimal</b>	0	1	2	3	4	5	6	7

<b>dezimal</b>	8	9	10	11	12	13	14	15
<b>binär</b>	1000	1001	1010	1011	1100	1101	1110	1111
<b>hexadezimal</b>	8	9	A	B	C	D	E	F

<b>Wert</b>	127	128	255	256	1k	4k	16k	64k
<b>dezimal</b>	127	128	255	256	1 024	4 096	16 384	65 536
<b>hexadezimal</b>	0x7F	0x80	0xFF	0x0100	0x0400	0x1000	0x4000	0x1 0000
<b>Bereich</b>		0x00 .... 0x7F		0x00 .... 0xFF	0x0000 .... 0x03FF	0x0000 .... 0x0FFF	0x0000 .... 0x3FFF	0x0000 .... 0xFFFF



## Gebräuchliche Informationseinheiten

Bezeichnung	Wertebereich			KEIL C Datentyp
	Bits	unsigned	signed	
in 8051 Entwicklungsumgebung				
Byte	8	0 .... 255	-128 ... +127	char
Wort	16	0 .... 64k-1	-32k ... +32k -1	int
(Doppelwort)	32	0 .... 4M-1	-2M ... +2M -1	

## IT Größenangaben

Kilo	Mega	Giga	Tera	Peta	Exa
$2^{10}$	$2^{20}$	$2^{30}$	$2^{40}$	$2^{50}$	$2^{60}$
$1024^1 \approx 10^3$	$1024^2 \approx 10^6$	$1024^3 \approx 10^9$	$1024^4 \approx 10^{12}$	$1024^5 \approx 10^{15}$	$1024^6 \approx 10^{18}$

### Beispiel: Umrechnung Anzahl Bits → Wertebereich

- Mit n Ziffern kann man Basis<sup>n</sup> unterschiedliche Werte darstellen : 0, 1 .... Basis<sup>n</sup> – 1; Anzahl = Basis<sup>n</sup> = '1' || n Nullen
  - Beispiel : Dezimalsystem mit n = 2 : **Werte** = 00, 01, ... , 99; **Anzahl** = 10<sup>2</sup> = 100
- Mathematische Basis : 1000 = 10<sup>3</sup> ≈ 2<sup>10</sup> = 1024 = 1k → 10<sup>3x</sup> ≈ 2<sup>10x</sup>
- Beispiel 1 : Anzahl = 2<sup>12</sup> = 1 0000 0000 0000B (binäre '1' und 12 binäre Nullen)
  - 2<sup>12</sup> By = 2<sup>2+10</sup> By = 2<sup>2</sup> \* 2<sup>10</sup> By ≈ 2<sup>2</sup> \* 10<sup>3</sup> By = 4 kB
- Beispiel 2 : 4 GB = 4 \* 10<sup>9</sup> By
  - 4 GB = 4 \* 10<sup>9</sup> By = 4 \* 10<sup>3</sup> \* 10<sup>3</sup> \* 10<sup>3</sup> By ≈ 2<sup>2</sup> \* 2<sup>10</sup> \* 2<sup>10</sup> \* 2<sup>10</sup> By = 2<sup>32</sup> By

# Gleitkommazahlen (I)

$$\begin{array}{c}
 \text{EVZ} \\
 \downarrow \\
 \text{Dezimale Gleitkommazahl - Beispiel : } + 123,456789 \cdot 10^{+14} \\
 \begin{array}{ccc}
 \uparrow & \uparrow & \uparrow \\
 \text{VZ} & \text{Mantisse} & \text{Exponent}
 \end{array}
 \end{array}$$

- Komma um eine Position nach links : Zahl wird um Faktor 10 kleiner.
- Exponent um 1 vergrößern : Zahl wird um Faktor 10 größer

$$\text{Normierte Darstellung : } + 0,123456789 \cdot 10^{+17}$$

$$\begin{array}{c}
 \text{Binäre Gleitkommazahl - Beispiel : } + 1011,110010011 \cdot 2^{-1010} \\
 \uparrow \\
 1 \cdot 2^{-1} = \frac{1}{2}
 \end{array}$$

# Gleitkommazahlen (II)

## Umwandlung in Speicherformat

- Beispiel:  $+1011,110010011 \cdot 2^{-1010}$

1. Normierung der Mantisse : Verschieben des Kommas hinter die erste '1'.

- Beispiel:  $+1,011110010011 \cdot 2^{-0111}$
- '1' vor dem Komma muss nicht gespeichert werden (spart ein Bit)

2. Ermittlung der Charakteristik :

$$\text{Charakteristik} = \text{Exponent} - \text{Exponent}_{\min}$$

$$\begin{array}{ccccc} \text{Exponent}_{\min} & \leq & \text{Exponent} & \leq & \text{Exponent}_{\max} \\ \downarrow & & \downarrow & & \downarrow \\ 0 & \leq & \text{Charakteristik} & \leq & \text{Charakteristik}_{\max} \end{array}$$

- Nur positive Charakteristikwerte - vermeidet Vorzeichen für den Exponenten

	Exponent	8 Bit Charakteristik
Min	-128	0
	0	128
Max	+127	255

# Gleitkommazahlen (III)

## Darstellung im Speicher

- IEEE 754 Short Precision : **32** Bits

VZ | Characteristic | Nachkommastellen der Mantisse  
 1            8                                  23 Bits

Beispiel : 0    0111 1001    0111 1001 0011 0000 0000 000

- VZ = 0 : positive Gleitkommazahl
  - VZ = 1 : negative Gleitkommazahl
- 
- IEEE 754 Double Precision : **64** Bits
    - Charakteristik : 11 Bits
    - Mantisse : 52 Bits

# Binär codierte Dezimalzahlen - Binary Coded Decimal (BCD)

## Ungepackte BCD-Zahlen (unpacked BCD)

- Die Dezimalziffern '0' ... '9' werden durch die binären Zahlen '0000 0000' bis '0000 1001' dargestellt.
  - Die Bitkombinationen '0000 1010' bis '1111 1111' sind ungültig.
- Pro Ziffer einer ungepackten Dezimalzahl wird also ein Byte benötigt.

## Gepackte BCD-Zahlen (packed BCD)

- Die Dezimalziffern '0' ... '9' werden durch die binären Zahlen '0000' bis '1001' dargestellt.
- Bei **gepackten BCD Zahlen** (packed BCD) werden zwei Dezimalziffern in einem Byte gespeichert
  - Die erste Ziffer in Bits 7 ... 4,
  - Die zweite Ziffer in Bits 3 ... 0.
  - '0011 1000'B entspricht z.B. der dezimalen Zahl '38'.
  - Da ein Halbbyte 16 Werte annehmen kann, eine Dezimalziffer allerdings nur 10, gibt es 6 nicht verwendete Bitkombinationen
    - '1010'B ... '1111'B (0xA ... 0xF)
    - Sie werden auch **Pseudotetraden** genannt.