

DS-Systeme Kapitel 4

Data Representation

Whole Numbers



Inhaltsverzeichnis

Thema	Seite
Motivation	3
Additionssysteme	5
Stellenwertsysteme	7
Zahlensysteme	8
Umrechnung dezimal \Leftrightarrow dual	9
Umrechnung dezimal \Leftrightarrow oktal	10
Umrechnung dezimal \Leftrightarrow hexadezimal	11
Umrechnung dual \Leftrightarrow oktal - Dualtriaden	12
Umrechnung dual \Leftrightarrow hexadezimal - Dualtetraden	13
Darstellung von Zahlen in C	14
Darstellung vorzeichenbehafteter Ganzzahl-Datentypen	17
Darstellung vorzeichenloser Datentypen	20
Multibyte-Darstellungen - Little / Big Endian	21

Motivation (Darstellung von Zahlen und Zeichen)

Ein Softwarefehler war die Ursache für den Absturz der Ariane 5 am 4. Juni 1996.

Unter anderem war ein Integer-Overflow die Ursache. Zitat aus dem Originalbericht:

"The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer."

<http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html>

Es gab einen Schaden in Höhe von 370.000.000 US-Dollar.



Motivation

Verständnis entwickeln für Phänomene folgender Art:

```
int    x = 200 * 300 * 400 * 500;    //-> x = -884 901 888  
  
float  y1 = (3.14 + 1e20) - 1e20;    //-> y1 = 0, aber  
float  y2 = 3.14 + (1e20 - 1e20);    //-> y2 = 3.14
```

Additionssysteme

Ein Additionssystem ist ein Zahlensystem, bei dem sich der Wert einer Zahl durch Addieren der Werte ihrer Ziffern errechnet.

Beispiele:

Strichliste:

Dabei stellt jeder Strich eine 1 dar. Die Gesamtzahl ist also die Anzahl der Striche. Z.B.: 3 = |||, 13 = ~~||||~~ ~~||||~~ |||

Römische Zahlen:

Symbole: **I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000.**

Z.B. hatten die Zahlen **IXIVDII** (= 1 + 10 + 1 + 5 + 500 + 1 + 1 = 519) früher die gleiche Bedeutung wie **DXVIII**. D.h. die unterschiedlichen Symbole wurden nach ihrer Wertigkeit addiert, aber die Reihenfolge spielte keine Rolle.

Additionssysteme

Römische Zahlen:

Symbole: **I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000.**

Regeln heute:

- a)römische Ziffern I, X, C, M (Grundzahlen) - diese dürfen maximal dreimal direkt nebeneinander stehen
- b)römische Ziffern V, L, D (Zwischenzahlen)- diese dürfen nur einmal nebeneinander stehen
- c)Die Grundzahlen I, X, C dürfen nur von einem der beiden nächstgrößeren Zahlzeichen subtrahiert werden.

Erlaubt sind z.B. IV = 4, XL = 40, CD = 400
verboten sind z.B. IC = 99, VD = 495, XM = 990

Aufgabe:

Geben Sie nach diesem System die römischen Zahlen für die Dezimalzahlen 447, 519 und 999 an.

Stellenwertsysteme

Ein Stellenwertsystem, Positionssystem oder polyadisches Zahlensystem ist ein Zahlensystem, bei dem die (additive) Wertigkeit eines Symbols von seiner Position, der Stelle, abhängt.

Unter der Annahme eines endlichen Vorrats an Symbolen **b** (b = Basis) (meist Ziffern genannt, z.B. [0 ... 9] im Zehnersystem) hängt die Anzahl der erforderlichen Stellen logarithmisch von der Größe der dargestellten Zahl ab.

Bei den wichtigen ganzzahligen Systemen ist der Wert der dargestellten Zahl gleich der Summe der Produkte des jeweiligen Ziffernwertes mit seinem Stellenwert, also ein Polynom in **b** mit den Werten der Ziffern als Koeffizienten.

z.B.: $519_{10} = 5 * 100 + 1 * 10 + 9 * 1 = 5 * 10^2 + 1 * 10^1 + 9 * 10^0$
(mit den **Stellenwerten** 100, 10 und 1, bzw. 10^2 , 10^1 und 10^0)

Stellenwertsysteme und ihre **b-adische** Darstellung:

Basis b = 2, Ziffernvorrat {0, 1}

Basis b = 8, Ziffernvorrat {0, 1, 2, 3, 4, 5, 6, 7}

Basis b = 10, Ziffernvorrat {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Basis b = 16, Ziffernvorrat {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Natürliche Zahlen werden in der b-adischen Darstellung durch eine endliche Folge von Ziffern $a_{n-1} \dots a_2 a_1 a_0$ (n = Anzahl der Ziffern) in der Form

$$Z = \sum_{i=0}^{n-1} a_i b^i = a_0 b^0 + a_1 b^1 + a_2 b^2 + \dots + a_{n-1} b^{n-1} \quad \text{dargestellt.}$$

Zahlensysteme

Gegenüberstellung von Zahlensystemen mit unterschiedlichen Basen:

Zahlensystem	Basis
dual	2
oktal	8
dezimal	10
hexadezimal	16

Dualzahlen, die man aus der dezimalen Formel $2^i - 1, (i \in \mathbb{N} \setminus \{0\})$ berechnen kann bestehen nur aus Einsen.

Zweier-
potenzen

$2^1 - 1$ 2^0

2^1

$2^2 - 1$

2^2

$2^3 - 1$

2^3

$2^4 - 1$

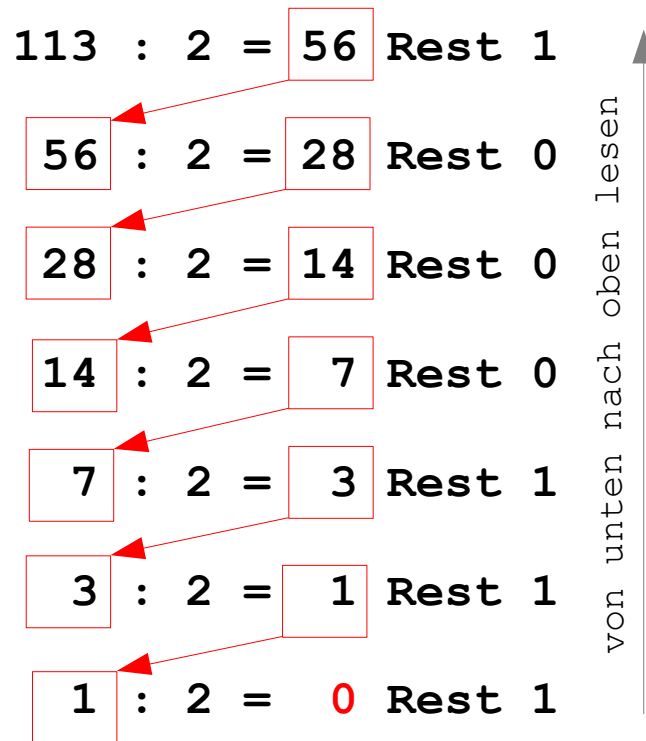
2^4

dual	oktal	dezimal	hexadezimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	a
1011	13	11	b
1100	14	12	c
1101	15	13	d
1110	16	14	e
1111	17	15	f
10000	20	16	10

Umrechnung dezimal \Leftrightarrow dual

dezimal \rightarrow dual

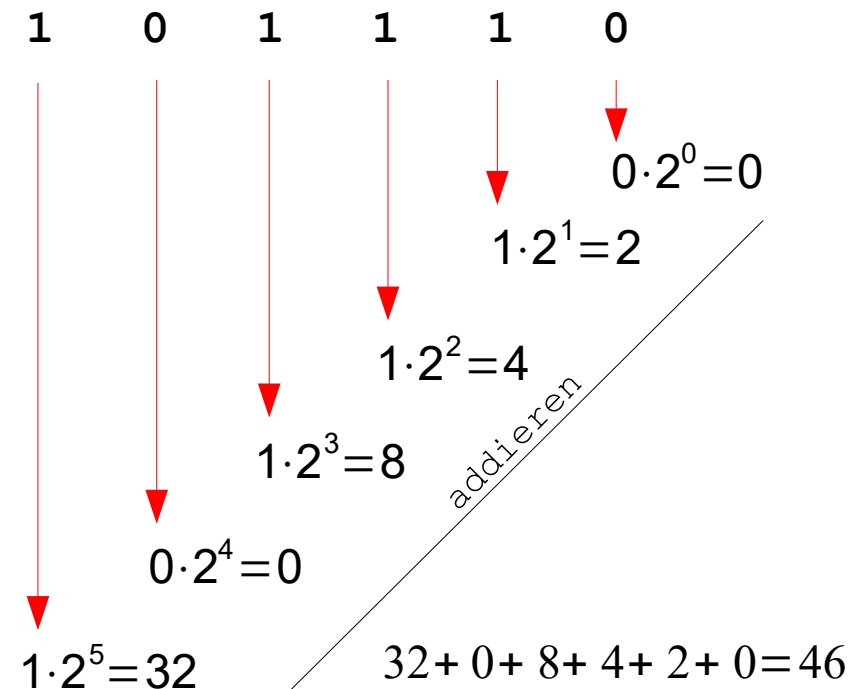
Beispiel: $113_{10} = ?_2$



$$113_{10} = 1110001_2$$

dual \rightarrow dezimal

Beispiel: $101110_2 = ?_{10}$

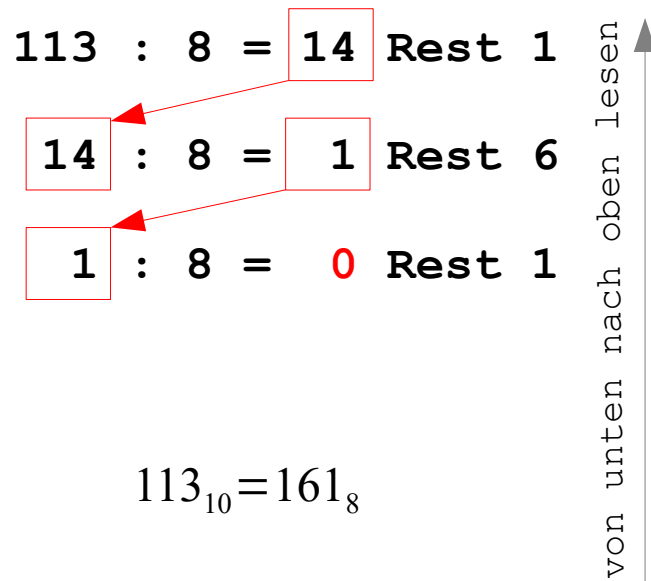


$$101110_2 = 46_{10}$$

Umrechnung dezimal \Leftrightarrow oktal

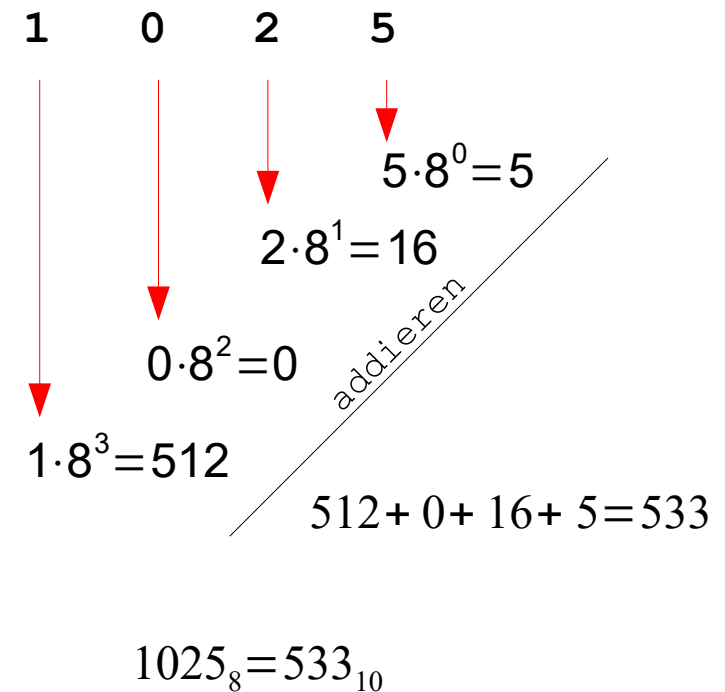
dezimal \rightarrow oktal

Beispiel: $113_{10} = ?_8$



oktal \rightarrow dezimal

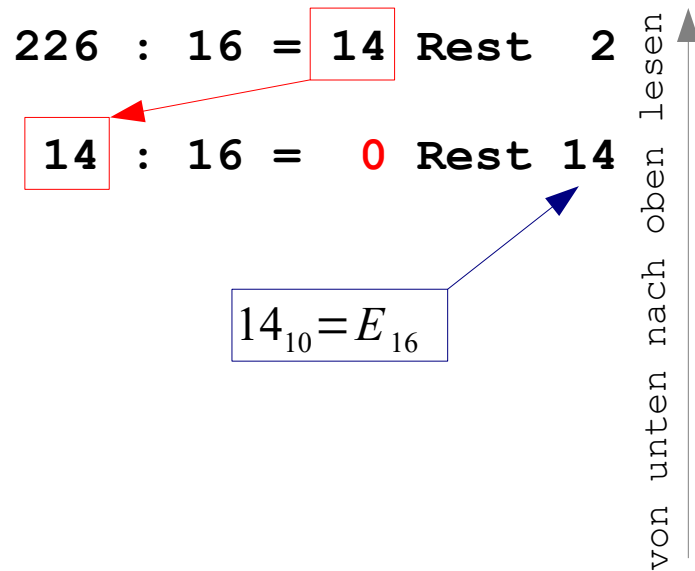
Beispiel: $1025_8 = ?_{10}$



Umrechnung dezimal \Leftrightarrow hexadezimal

dezimal \rightarrow hexadezimal

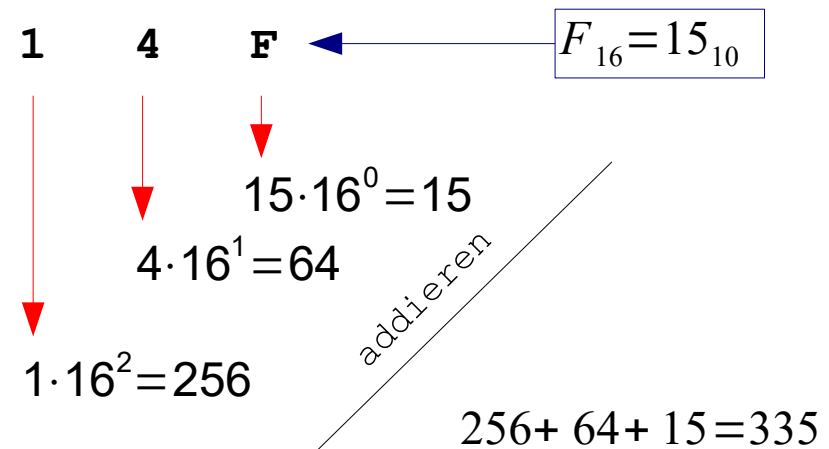
Beispiel: $226_{10} = ?_{16}$



$$226_{10} = E2_{16}$$

hexadezimal \rightarrow dezimal

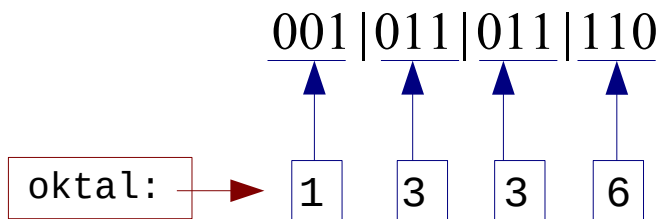
Beispiel: $14F_{16} = ?_{10}$



$$14F_{16} = 335_{10}$$

Umrechnung dual \Leftrightarrow oktal mittels Dualtriatendual \rightarrow oktalBeispiel: $1011011110_2 = ?_8$

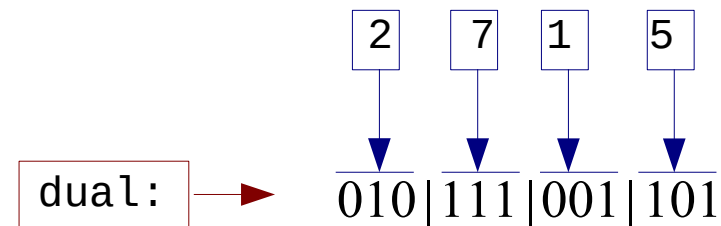
Unterteilung der Dualzahl
von rechts in Dreiergrup-
pen (**Dualtriaten**):



$$1011011110_2 = 1336_8$$

oktal \rightarrow dualBeispiel: $2715_8 = ?_2$

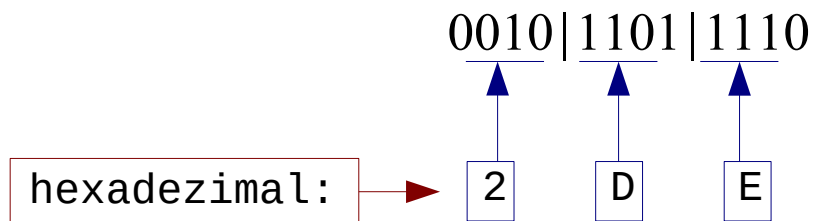
Umwandlung jeder einzelnen
Ziffer in eine dreistellige
Dualzahl (**Dualtriade**):



$$2715_8 = 10111001101_2$$

Umrechnung dual \Leftrightarrow hexadezimal mittels Dualtetradendual \rightarrow hexadezimalBeispiel: $1011011110_2 = ?_{16}$

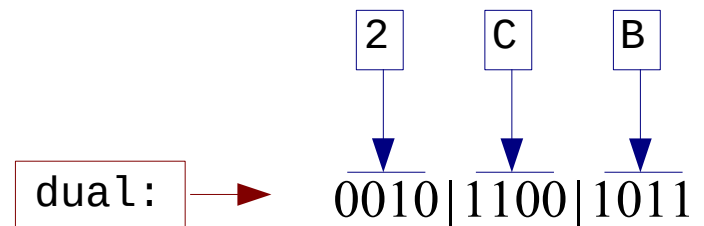
Unterteilung der Dualzahl
von rechts in Vierergrup-
pen (**Dualtetraden**):



$$1011011110_2 = 2DE_{16}$$

hexadezimal \rightarrow dualBeispiel: $2CB_{16} = ?_2$

Umwandlung jeder einzelnen
Ziffer in eine vierstelli-
ge Dualzahl (**Dualtetrade**):



$$2CB_{16} = 1011001011_2$$

Darstellung von Zahlen in C

Zur Zahlendarstellung in C gehören Zeichen-, Ganzzahl- und Gleitkommatypen:

Typen	vorzeichenbehaftet	vorzeichenlos
Zeichentypen	signed char	unsigned char
Ganzzahltypen	signed char signed short int signed int signed long int signed long long int	unsigned char unsigned short int unsigned int unsigned long int unsigned long long int
Gleitkommatypen	float double long double	

integrale
Datentypen

Die Schlüsselwörter **signed** und **int**, die rot gekennzeichnet sind, kann man bei den entsprechenden Datentypen auch weglassen.

Darstellung von Zahlen in C

In der folgenden Tabelle sind alle *elementaren Datentypen* mit ihrem Speicherverbrauch und ihrem Wertebereich bzw. ihrer Belegung dargestellt wie sie üblicherweise auf den meisten Betriebssystemen implementiert sind.

Datentyp	Bytes	Belegung bzw. Wertebereich
char	1	$(-2^7, 2^7 - 1) = (-128, 127)$
unsigned char	1	$(0, 2^8 - 1) = (0, 255)$
short	2	$(-2^{15}, 2^{15} - 1) = (-32768, 32767)$
unsigned short	2	$(0, 2^{16}-1) = (0, 65535)$
int	4	$(-2^{31}, 2^{31} - 1) = (-2147483648, 2147483647)$
unsigned int	4	$(0, 2^{32}-1) = (0, 4294967295)$
long	4	$(-2^{31}, 2^{31} - 1) = (-2147483648, 2147483647)$
unsigned long	4	$(0, 2^{32}-1) = (0, 4294967295)$
long long	8	$(-2^{63}, 2^{63} - 1)$ $(-9223372036854775808, 9223372036854775807)$
unsigned long long	8	$(0, 2^{64}-1) = (0, 18446744073709551615)$
float	4	$\pm(1.17549435\text{E-}38, 3.40282347\text{E+}38)$
double	8	$\pm(2.225073858507201\text{E-}308, 1.79769313486231570\text{E+}308)$
long double	16	systemabhängig

Darstellung von Zahlen in C

Die gleiche Tabelle noch einmal mit den Konstanten aus den Header-Dateien `<limits.h>` und `<floats.h>`.

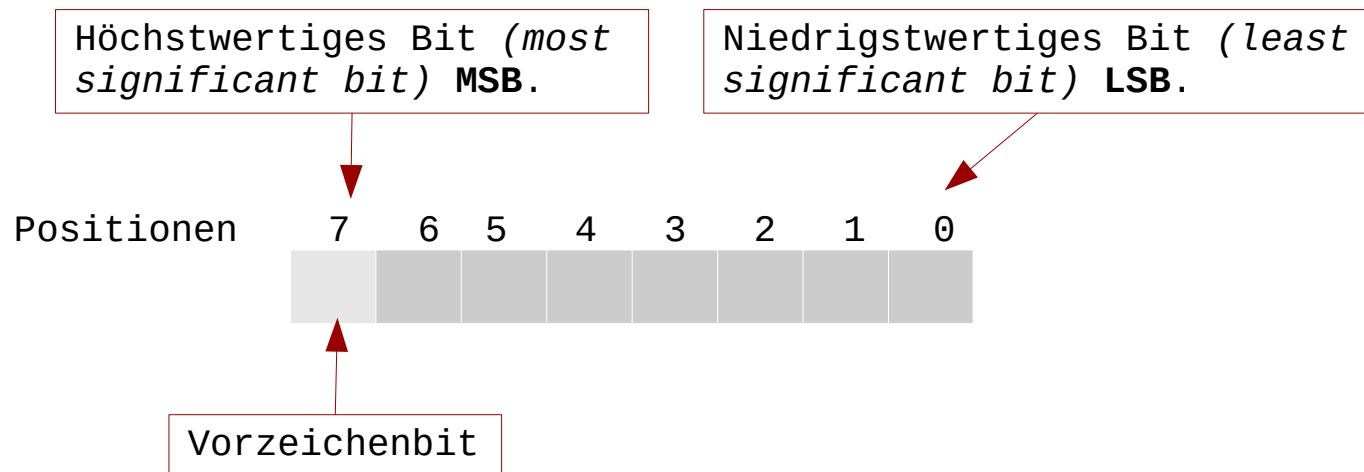
`<limits.h>`

Datentyp	Bytes	Wertebereich
char	1	(CHR_MIN, CHR_MAX)
unsigned char	1	(0, UCHAR_MAX)
short	2	(SHRT_MIN, SHRT_MAX)
unsigned short	2	(0, USHRT_MAX)
int	4	(INT_MIN, INT_MAX)
unsigned int	4	(0, UINT_MAX)
long	4	(LONG_MIN, LONG_MAX)
unsigned long	4	(0, ULONG_MAX)
long long	8	(LLONG_MIN, LLONG_MAX)
unsigned long long	8	(0, ULLONG_MAX)
float	4	(FLT_MIN, FLT_MAX)
double	8	(DBL_MIN, DBL_MAX)
long double	16	(LDBL_MIN, LDBL_MAX)

`<float.h>`

Darstellung von vorzeichenbehafteter Ganzzahltypen (signed)

Aufbau eines Ganzzahltyps gezeigt am Datentyp `char`.



Der prinzipielle Aufbau ist bei allen Ganzzahltypen gleich.

Ganzzahltypen nennt man auch **integrale Datentypen**.

Beispiele:

$$113_{10} = 01110001_2$$



0 bedeutet positiv:
Zahl ≥ 0

$$-113_{10} = 10001111_2$$



1 bedeutet negativ:
Zahl < 0

Achtung:

Hier hat sich nicht nur das Vorzeichenbit geändert.

Darstellung negativer Ganzzahlen - Zweierkomplement

Darstellung negativer Zahlen unter Verwendung des **Zweierkomplements** - gezeigt am Datentyp **short**.

Um eine positiven Ganzzahl in eine negative umzuwandeln bzw. umgekehrt, invertiert man zunächst alle Bits und addiert anschließend 1 hinzu.

Beispiel: Umwandlung positiv → negativ

$$46_{10} = 000000000101110_2$$

0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Schritt 1: Alle Bits (einschließlich Vorzeichenbit) invertieren.

1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Schritt 2: 1 hinzuaddieren + 1

$$-46_{10} = 111111111010010_2$$

1	1	1	1	1	1	1	1	1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑
Dadurch ergibt sich **automatisch** im Vorzeichenbit die **1**.

Darstellung negativer Ganzzahlen - Zweierkomplement

Darstellung negativer Zahlen unter Verwendung des **Zweierkomplements** - gezeigt am Datentyp **short**.

Um eine positiven Ganzzahl in eine negative umzuwandeln bzw. umgekehrt, invertiert man zunächst alle Bits und addiert anschließend 1 hinzu.

Beispiel: Umwandlung negativ → positiv

$$-46_{10} = 111111111010010_2$$

1	1	1	1	1	1	1	1	1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Schritt 1: Alle Bits (einschließlich Vorzeichenbit) invertieren.

0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Schritt 2: 1 hinzuaddieren

+	1
---	---

$$46_{10} = 000000000101110_2$$

0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Dadurch ergibt sich **automatisch** im Vorzeichenbit die **0**.

Darstellung vorzeichenloser Ganzzahl-Datentypen (**unsigned**)

Ganzzahl-Datentypen besitzen, wenn nichts anderes angegeben ist, immer den Qualifizierer **signed**. Das bedeutet, dass ihr Wertebereich sowohl negative als auch positive Zahlen enthält.

Dabei ist das **Most Significant Bit (MSB)** immer für das Vorzeichen reserviert. **MSB = 0** bedeutet positive Zahl und **MSB = 1** eine negative Zahl.

Der Wertebereich liegt immer in

$$[-2^{n-1}, 2^{n-1}-1], n = \text{Anzahl Bits des Datentyps}$$

Beispiel: Datentyp **signed char**

Wertebereich: $[-2^7, 2^7-1] = [-128, 127], \text{\#Bits} = 8$

Bei vorzeichenlosen Ganzzahltypen wird der Qualifizierer **unsigned** immer angegeben. Das MSB ist hier nicht für ein Vorzeichen reserviert, da der Zahlenbereich nur positive Werte annimmt.

Wertebereich:

$$[0, 2^n-1], n = \text{Anzahl Bits des Datentyps}$$

Beispiel: Datentyp **unsigned char**

Wertebereich: $[0, 2^8-1] = [0, 255], \text{\#Bits} = 8$

Multibyte-Darstellungen - Little / Big Endian

Computersysteme verwenden typischerweise **Bytes** als kleinste (logisch) adressierbare Einheit.

Praktisch werden Daten in größeren Blöcken

32-Bit-Plattform 4 Byte (32 Bit)

64-Bit Plattform 8 Byte (64 Bit)

adressiert.

Solche Multibyte-Datenblöcke werden im Speicher als zusammenhängende Bytesequenz gespeichert, die durch die kleinste Adresse adressiert wird.

Beispiel:

int **n** hat Adresse **0x100**

==> **n** ist gespeichert in Adresse **0x100**, **0x101**, **0x102** und **0x103**

Wie allerdings die einzelnen Bytes in den Adressen **0x100** bis **0x103** angeordnet sind, ist architektur- bzw. herstellerabhängig.

Dies führt zur Unterscheidung von **Little Endian** und **Big Endian**.

Multibyte-Darstellungen - Little / Big Endian

Beispiel:

```
int n = 0x01234567; // &n = 0x100
```

Variante **Big Endian**



Variante Little Endian



Big Endian:

Das höchstwertige Byte befindet sich an der niedrigsten Adresse. Beispiele: PPC (IBM), SPARC (SUN / ORACLE).

Little Endian:

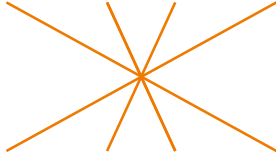
Das niederwertigste Byte befindet sich an der niedrigsten Adresse. Beispiele: x86 (Intel, AMD).

Multibyte-Darstellungen - Little / Big EndianBeispiel:

```
int n = 0x01234567; // &n = 0x100
```

Die Endianness ist insbesondere auch bei Netzwerkübertragungen zwischen Systemen mit unterschiedlicher Endianness wichtig.

Die Berücksichtigung von Little / Big Endian Darstellung ist insbesondere beim *Disassemblieren (Befehl **objdump -d**) wichtig:

	Maschinenbefehl	Assemblerbefehl
80483bd:	01 05 64 94 04 08	add %eax, 0x8049464
		
	08 04 94 64 = 8049464	

*Ein Disassembler ist ein Computerprogramm, das die binär kodierte Maschinensprache eines ausführbaren Programmes in eine für Menschen lesbarere Assemblersprache umwandelt. Im Gegensatz zu einem Assembler, der Assemblersprache in Maschinencode konvertiert, hat der Disassembler die Aufgabe, ausführbaren Maschinencode in Assemblersprache zu übersetzen.