Integer Division , Modulo Operator

→ Integer Division : Normales "geteilt durch" aber es wird immer gegen 0 gerundet.

→ Modulo : Rest, der übrig bleibt nach integer division.

Es gilt:

$$(a / b) * b + a \,\%\, b = a$$

$\Leftrightarrow a \,\%\, b = a - (a/b) \cdot b$ → kann hilfreich sein beim verstehen von Modulo operationen

Bsp. $\frac{7}{3} = 2.\overline{3} \xrightarrow{C++} 2$ (weil gegen Null gerundet wird.)

$\frac{15}{4} = 3.75 \longrightarrow 3$

$\frac{16}{4} = 4 \longrightarrow 4$

modulo: $7 \,\%\, 3 \rightarrow 6$ rest 1

$15 \,\%\, 4 \rightarrow 12$ rest 3

$16 \,\%\, 4 \rightarrow 4$ rest 0

(zusätzlich: Negative Zahlen)

Einfache Regel: immer das Vorzeichen vom linken Operand.

Eine Zahl ist negativ

```
std::cout << (-7 % 3) << std::endl;   == -1
std::cout << (7 % -3) << std::endl;   == -1
```

(-7/3) => -2
-2 * 3 => -6
so a%b => -1 weil $-7 - (-7/3) \cdot 3 = -7 - (-2) \cdot 3 = -7 - (-6) = -7 + 6 = -1$

(7/-3) => -2
-2 * -3 => 6
so a%b => 1 weil $7 - 6 = 1$

beide sind negativ

$-7 \,\%\, -3 == -1$

weil $-7/-3 \rightarrow 2$

$2 * -3 \rightarrow -6$

$\rightarrow -7 - (-7/-3) * -3 = -7 - (2) * -3 = -7 - (-6) = -7 + 6 = -1$

Binary Representation and unsigned integers

→ Alle Zahlen werden in einer Basis dargestellt:
Wenn wir 123 sagen, meinen wir
$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$
⇒ Die Stellen stehen für Exponenten der Basis

	1	2	3
Basis:	$1 \cdot 10^2$	$2 \cdot 10^1$	$3 \cdot 10^0$
Exponent:	2	1	0

⇒ im binären System genau gleich!

Bsp.

1	1	1	1	0	1	
5	4	3	2	1	0	
2^5	2^4	2^3	2^2	2^1	2^0	
32	16	8	4	2	1	
32 +	16 +	8 +	4 +		+ 1	= 61

→ oft schreibt man binäre Zahlen mit einem "0b" davor also hier 0b111101

Negative (ganze) Zahlen

→ negative Zahlen sollen erfüllen $x + (-x) = 0$.
⇒ wir müssen nur etwas finden, welche diese Regel erfüllt.

Bsp. 4-bit binäre Zahlen ****

1 → 0001

⇒ d.h. wir wollen

$$\begin{array}{r} 0001 \\ +\,xxxx \\ \hline 0000 \end{array}$$

→ Wir wissen 0b1 + 0b1 = 0b10
also:

$$\begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

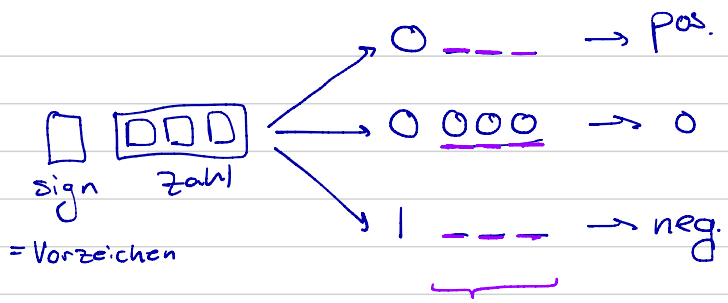
Darum

$$\begin{array}{r|l} 1\,1\,1\,1 & \\ 0001 & \\ +\,1111 & \\ \hline 10000 & \end{array}$$

diese 1 kann mit 4-bits nicht dargestellt werden d.h. die Zahl die wir kriegen ist 0000 wie gewünscht.

Regel um $-x$ zu finden:
1) $|-x|$, also x binär
2) alle bits flippen (0 wird 1, 1 wird 0)
3) 1 addieren

→ Wir haben 4 bits:



⇒ mit 3 bits kann man 8 (pos.) Zahlen darstellen (0, 1, 2, 3, 4, 5, 6, 7) d.h. da eine davon 0 ist, ist 7 die grösste pos. Zahl.

⇒ für negative Zahlen (sign bit = 1) haben wir wieder 3 bits d.h. 8 Zahlen aber wir müssen keine "-0" machen d.h. -8 ist unsere betragsmässig grösste neg. Zahl.