Bitoperationen [=Java]

- Bitweise Verknüpfung von Ganzzahltypen, kommutativ

&	bitweises „Und“ (Bit-Schnittmenge)	$1 \,\&\, 1 \rightarrow 1$ $1 \,\&\, 0 \rightarrow 0$ $0 \,\&\, 0 \rightarrow 0$
|	bitweises „Oder“ (Bit-Vereinigungsmenge)	$1 \mid 1 \rightarrow 1$ $1 \mid 0 \rightarrow 1$ $0 \mid 0 \rightarrow 0$
∧	bitweises „Exklusiv-Oder“ (Bit-Antivalenz)	$1 \wedge 1 \rightarrow 0$ $1 \wedge 0 \rightarrow 1$ $0 \wedge 0 \rightarrow 0$
~	bitweise Inversion (Einerkomplement, unär)	$\sim 1 \rightarrow 0$ $\sim 0 \rightarrow 1$

Bitoperationen (Forts.) [=Java]

- Schiebeoperationen auf Ganzzahltypen, nicht kommutativ

<<	bitweises Linksschieben (rechts werden 0-Bits „nachgefüllt“)
>>	bitweises Rechtsschieben (links werden 0-Bits „nachgefüllt“)

- Beispiele (x sei vom Typ `uint8_t`)

Bit#	7	6	5	4	3	2	1	0	
x=156	1	0	0	1	1	1	0	0	0x9c
~x	0	1	1	0	0	0	1	1	0x63
7	0	0	0	0	0	1	1	1	0x07
x | 7	1	0	0	1	1	1	1	1	0x9f
x & 7	0	0	0	0	0	1	0	0	0x04
x ^ 7	1	0	0	1	1	0	1	1	0x9B
x << 2	0	1	1	1	0	0	0	0	0x70
x >> 1	0	1	0	0	1	1	1	0	0x4e

Bitoperationen – Anwendung

- Durch Verknüpfung lassen sich gezielt einzelne Bits setzen/löschen

Bit#	7	6	5	4	3	2	1	0	
PORTD	?	?	?	?	?	?	?	?	Bit 7 soll verändert werden, die anderen Bits jedoch erhalten bleiben!
0x80	1	0	0	0	0	0	0	0	Setzen eines Bits durch **Ver-odern** mit Maske, in der nur das Zielbit 1 ist
PORTD |= 0x80	1	?	?	?	?	?	?	?	
~0x80	0	1	1	1	1	1	1	1	Löschen eines Bits durch **Ver-unden** mit Maske, in der nur das Zielbit 0 ist
PORTD &= ~0x80	0	?	?	?	?	?	?	?	
0x08	0	0	0	0	1	0	0	0	Invertieren eines Bits durch **Ver-xodern** mit Maske, in der nur das Zielbit 1 ist
PORTD ^= 0x08	?	?	?	?	¿	?	?	?	

Bitoperationen – Anwendung (Forts.)

- Bitmasken werden gerne als Hexadezimal-Literale angegeben

Bit#	7	6	5	4	3	2	1	0	
0x8f	1	0	0	0	1	1	1	1	Jede Hex-Ziffer repräsentiert genau ein Halb-Byte (*Nibble*) ⇝ Verständlichkeit
	8				f				

- Für „Dezimal-Denker“ bietet sich die Linksschiebe-Operation an

```
PORTD |= (1<<7);    // set bit 7:  1<<7 --> 10000000
PORTD &= ~(1<<7);   // mask bit 7: ~(1<<7) --> 01111111
```

- Zusammen mit der Oder-Operation auch für komplexere Masken

```
#include <led.h>
void main() {
  uint8_t mask = (1<<RED0) | (1<<RED1);

  sb_led_set_all_leds (mask);

  while(1) ;
}
```

⇝