3 Bitoperatoren

Oft wird die Sprache C im hardwarenahen Bereich eingesetzt. Dadurch werden zusätzlich zu den üblichen Operatoren gerade die Bit-Operatoren häufig verwendet.

Links Schieben <<

`3 << 2` bedeutet 3 um 2 Bit Positionen nach Links schieben. 3dez (= 11bin) verschoben ergibt **1100bin** (= 12dez). Jede Verschiebung um 1 Bit nach Links entspricht einer Multiplikation mit 2 (3x2x2 = 12).

Rechts Schieben >>

`7 >> 2` bedeutet 7 um 2 Bit Positionen nach Links schieben. 7dez (= 111bin) verschoben ergibt **1bin** (= 1dez). Jede Verschiebung um 1 Bit nach Rechts entspricht einer Division durch 2 (7 / 2 / 2 = 1).

Bitweise Verundung &

`7 & 5` (**111bin** & **101bin**) ergibt **101bin**. 1te Stelle der 1ten Zahl mit 1ter Stelle der 2ten Zahl verundet usf.

Bitweise Veroderung |

`0x34 | 0x22` (**0011 0100bin** | **0010 0010bin**) ergibt **0011 0110bin**. 1te Stelle der 1ten Zahl mit 1ter Stelle der 2ten Zahl verodert usf.

Bitweise Exor ^

`0x34 ^ 0x22` (**0110bin** ^ **0010bin**) ergibt **0100bin**. 1te Stelle der 1ten Zahl mit 1ter Stelle der 2ten Zahl verexort usf.

Bitweise Negation ~

`~0x22` (**~0010 0010bin**) ergibt **1101 1101bin**. An jeder Stelle wird aus 1 -> 0 und aus 0 -> 1.

Typische Aufgaben:

- es soll das nte Bit eines Registers r auf 1 gesetzt werden. Die verbleibenden Stellen sollen ihren Wert beibehalten.
  `r | (1 << (n-1))` Eine 1 wird um (n-1) Stellen nach Links verschoben --> ntes Bit ist 1. Bitweise Veroderung mit Registerwert setzt das nte Bit auf 1.

- es soll das nte Bit eines Registers r invertiert gesetzt werden. Die verbleibenden Stellen sollen ihren Wert beibehalten.
  `r ^ (1 << (n-1))` Eine 1 wird um (n-1) Stellen nach Links verschoben --> ntes Bit ist 1. Exor mit Registerwert: an allen Stellen von r in denen eine 0 steht bleibt eine 0 (0^0). An Stellen an denen eine 1 steht bleibt eine 1 (1^0). An der Stelle an der der Wert auf 0 gesetzt werden soll wird für r=1 mit 1^1=0 und für r=0 mit 0^1=1.

- es soll das nte Bit eines Registers r auf 0 gesetzt werden. Die verbleibenden Stellen sollen ihren Wert beibehalten.
  `r & ~(1 << (n-1))` Eine 1 wird um (n-1) Stellen nach Links verschoben --> ntes Bit ist 1. Durch Bitweise invertierung werden sämtliche Werte auf 1 gesetzt bis auf die nte Stelle. Verundung mit Registerwert: an allen Stellen von r wird der Wert beibehalten bis auf die nte Stelle, dort wird eine 0 gesetzt.

Sollen entsprechend mehrere Bitpositionen auf einmal gesetzt oder rückgesetzt werden, dann sind entsprechend mehrere Bits zu setzen.

Boolsche Ausdrücke

In C wird auf logisch Wahr oder Falsch geprüft bei:

- Verzweigungen `if (expr) {...`
- Schleifen `when (expr) {...`, `for (i = 0; expr; i++) {...` ...

In C gibt's allerdings keinen Boolschen Datentyp. Dadurch ist zu klären was unter Wahr und Falsch zu verstehen ist:

- Falsch = 0
- Wahr = Nicht(Falsch)

Für C ist alles Wahr was ungleich 0 ist. Das wirkt vieleicht kompliziert ist aber logisch: 0 ist immer Falsch (FALSE), alles Andere (1, 10, 50 ...) ist somit Wahr. Wann spielt das eine Rolle:

```
if (a == TRUE) {                // mit z.B. #define TRUE 1
```

Diese Prüfung ist zu vermeiden, sie läßt sich leicht ersetzen:

```
if (a != FALSE) {               // mit z.B. #define FALSE 0 : alles was nicht 0 ist ist wahr
```

Sonst hat man keine reine 2-Wertigkeit sondern 0, 1 und alles Andere (undefiniert).

Fragen

- Setzen Sie das nte Bit der Variable reg auf 1. Belassen Sie sämtliche anderen Bits gleich.
- Setzen Sie das nte Bit der Variable reg auf 0. Belassen Sie sämtliche anderen Bits gleich.
- Setzen Sie das nte Bit der Variable reg auf 1. Setzen Sie sämtliche anderen Bits auf 0.
- Invertieren Sie den Wert des nten Bits der Variable reg (0->1 bzw 1->0) und belassen Sie sämtliche anderen Bitwerde gleich.
- Setzten Sie sämtliche Bits der Variable reg auf 1 ausgenommen das nte Bit, das soll den Wert 0 erhalten.
- Ermitteln Sie den Zustand des nten Bits der Variable reg.