Merkblatt : Wichtige bitweise Operationen

1. Gezieltes Löschen von einzelnen Bits mittels AND-Verknüpfung:

Aufgabe: Lösche alle Bits in Byte b (=35 *hex*) bis auf Bit Nr. 2, lasse dieses unverändert.
Lösung: AND-Verknüpfung mit der „Maske“ 0000 0100 *bin* = 04 *hex*

```
  b      0011 0101
(AND)    0000 0100
 ----------------
  =      0000 0100
 ================
```

Programmierung z.B. in „C“ :

```
unsigned char b, e;
b = 0x35;
e =  b & 0x04;    // liefert  e = 04 hex
```

Anwendung: z.B. für gezieltes Abfragen oder Löschen von Portpins.

2. Gezieltes Setzen von einzelnen Bits mittels OR-Verknüpfung:

Aufgabe: Setze in Byte b (=35 *hex*) die Bits Nr. 5 und 6 , lasse den Rest unverändert.
Lösung: OR-Verknüpfung mit der „Maske“ 0110 0000 *bin* = 60 *hex*

```
  b      0011 0101
 (OR)    0110 0000
 ----------------
  =      0111 0101
 ================
```

Programmierung z.B. in „C“ :

```
unsigned char b, e;
b = 0x35;
e =  b | 0x60;    // liefert  e = 75 hex
```

Anwendung: z.B. für gezieltes „Einschalten“ von Portpins.

3. Gezieltes Invertieren von einzelnen Bits mittels EXOR-Verknüpfung:

Aufgabe: Invertiere in Byte b (=35 *hex*) die Bits Nr. 0 und 1 , lasse den Rest unverändert.
Lösung: EXOR-Verknüpfung mit der „Maske“ 0000 0011 *bin* = 03 *hex*

```
  b      0011 0101
(EXOR)   0000 0011
 ----------------
  =      0011 0110
 ================
```

Programmierung z.B. in „C“ :

```
unsigned char b, e;
b = 0x35;
e =  b ^ 0x03;    // liefert  e = 36 hex
```

Anwendung: z.B. für gezieltes Umschalten („Toggeln“) von Portpins.

4. Integer-Division durch 2, 4, 8, etc. durch Rechtsschieben im Akku:

```
 37   0010 0101
 :4   00 00 1001  01  ->
 ----------------
 =9   0000 1001  01  Rest
 ================
```

Division durch 4 ≡ Rechtsschieben um 2 Stellen

Programmierung z.B. in „C“ :

```
unsigned char b, e;
b = 37;
e =  b >> 2;   // liefert  e = 9,
               //  Rest 1 geht verloren !
```

5. Multiplikation mal 2, 4, 8, etc. durch Linksschieben im Akku:

```
  5       0000 0101
 *8   000  0010 1 000  <-
-----------------
=40      0010 1000
=================
```

Multiplik. mit 8 ≡ Linksschieben um 3 Stellen

Programmierung z.B. in „C“ :

```
unsigned char b, e;
b = 5;
e =  b << 3;   // liefert  e = 40 ,  bei einem
//  Produkt > 255  wäre d. Ergebnis falsch !
```