## Bit-Operatoren

|  |  |
| --- | --- |
| **Bitoperator** | **Formatzeichen** |
| &, &= | Bitweise AND-Verknüpfung |
| |, |= | Bitweise OR-Verknüpfung |
| ^, ^= | Bitweise XOR |
| ~ | Bitweises Komplement |
| >>, >>= | Rechtsverschiebung |
| <<, <<= | Linksverschiebung |

### Bitweises UND &

Steht der &-Operator zwischen zwei Operanden, so handelt es sich um den bitweisen UND-Operator. Dieser ist leicht mit dem unären Adressoperator zu verwechseln.

Der Operator wird hauptsächlich dafür verwendet, einzelne Bits gezielt zu löschen.

char x= 3;

x= x & 1; // alles außer 1.Bit wird gelöscht!

### Bitweises ODER |

Mit dem bitweisen ODER-Operator können Sie gezielt zusätzliche Bits setzen. Verwendet wird dieser wie schon zuvor der bitweise UND-Operator:

char x = 1;

x = x|126; // x=127

### Bitweises XOR ^

Dieser exklusive ODER-Operator liefert nur dann eine 1 zurück, wenn beide Bits unterschiedlich sind. Er ist sehr gut geeignet, um Bits umzuschalten. Alle gesetzten Bits werden gelöscht und alle gelöschten gesetzt.

char x=20;

x = x^55; // x=35

### Bitweises Komplement ~

Der NOT-Operator (~) wirkt sich auf Zahlen so aus, dass er jedes einzelne Bit invertiert.

char x=20;

x=~x; /\* x= -21 \*/

### Linksverschiebung <<

Mit einer Linksverschiebung (<<) werden alle Bits einer Zahl um n Stellen nach links gerückt. Die rechts entstehenden Leerstellen werden mit 0 aufgefüllt ( Ergebnis bei – Wert compilerspezifisch). Bei einer Stelle wird ein \* mit 2, bei zwei mit 4, usw. durchgeführt.

### Rechtsverschiebung >>

Gegenstück zur Linksverschiebung. Hier wird durch 2^x dividiert.

### Beispiele

4. Bit auf “1” setzen

|  |
| --- |
| reg = reg | 0x08; bzw. reg = reg | 8; |

3. und 5. Bit auf “0” setzen

|  |
| --- |
| reg = reg & 0xEB;  reg = reg & ~0x |

6. Bit umschalten

|  |
| --- |
| reg = reg ^ 0x20; |

7. Bit abfragen, ob auf “1”

|  |
| --- |
| if(reg&64) |

Funktion – Parameter: Bitposition Register; Das angeg. Bit auf „1“ setzen

|  |
| --- |
| tmpreg = 0x01;  tmpred = tmpreg << pos;  reg = reg | tmpreg; |

### Rezept für Fortgeschrittene

int wert;

unsigned char reg;

scanf(„%d“,&wert);

reg=(char)wert;

Register als Bit-Folge ausgeben:

int hilf,i;

hilf=reg;

for (i=0; i<8; i++)

{

if (hilf & 128) printf(“1”);

else printf(“0”);

hilf=hilf <<1;

}

printf(“\n”);

int hilf,i;

hilf=reg;

for (i=0; i<8; i++)

{

printf(“%d”,hilf & 128);

hilf=hilf <<1;

}

printf(“\n”);