Zahlensysteme

Liebe Leserinnen und Leser! Zuerst müssen die Karten auf den Tisch. Bei dieser Beschreibung habe ich teilweise den KI-Assistent „Google Gemini“ zu Rate gezogen. Einige Textpassagen stammen eins zu eins vom KI-Assistent. Jedoch nicht die JavaScript Applikation! Dann geht bei mir der Spaß verloren. Lasse aber für die Zukunft auch hier die Tür offen.

Dezimal, Hexadezimal, Oktal und Binär sind verschiedene Zahlensysteme, die sich durch ihre Basis unterscheiden. Das Dezimalsystem (Basis 10) verwendet die Ziffern 0-9 und ist unser Alltagssystem. Das Binärsystem (Basis 2) nutzt nur 0 und 1 und ist die Grundlage der Computertechnik. Das Oktalsystem (Basis 8) verwendet 0-7. Das Hexadezimalsystem (Basis 16) nutzt 0-9 und die Buchstaben A-F und ist wegen der einfacheren Darstellung von Binärdaten in der Informatik wichtig.

Die **Hexadezimalzahl** ist ein Zahlensystem mit der Basis 16. Es verwendet die Ziffern 0-9 und die Buchstaben A-F, wobei A für 10, B für 11, C für 12, D für 13, E für 14 und F für 15 steht. Das Hexadezimalsystem findet breite Anwendung in der Informatik und digitalen Elektronik, da es eine kompaktere Darstellung von Binärzahlen ermöglicht. Da $2^4=16$ ist, kann jede Hexadezimalziffer genau vier Binärziffern (Bits) darstellen. Dies vereinfacht die Lesbarkeit und Handhabung von langen Binärketten.

Eine **Dezimalzahl** ist eine Zahl, die im Zehnersystem (Dezimalsystem) dargestellt wird und durch ein Komma vom ganzzahligen Teil getrennt wird. Sie besteht aus einem ganzzahligen Anteil vor dem Komma und einem Nachkommaanteil, der Bruchteile wie Zehntel, Hundertstel oder Tausendstel darstellt.

Eine **Oktalzahl** ist eine Zahl im Oktalsystem oder Achtersystem, das die Basis 8 verwendet und nur die Ziffern 0 bis 7 kennt. Sie ist nützlich, um Binärzahlen zu vereinfachen, da drei Binärziffern (Bits) einer einzigen Oktalziffer entsprechen. In der Programmierung werden Oktalzahlen oft mit einer **führenden Null (0)** gekennzeichnet, und sie finden auch Anwendung bei Unix-Dateizugriffsrechten und in der Flugsicherung.

Eine **Binärzahl** ist eine Zahl im binären Zahlensystem (Basis 2), die nur die Ziffern 0 und 1 verwendet, im Gegensatz zum Dezimalsystem mit den Ziffern 0 bis 9. Computer nutzen Binärzahlen, um alle Informationen als Abfolge von Nullen und Einsen (Bits) zu speichern und zu verarbeiten, da sie leicht in zwei Zustände (z. B. "an" oder "aus") umgesetzt werden können. Jede Null oder Eins wird als Bit bezeichnet, und acht Bits bilden ein Byte, das eine Einheit zur Speicherung von Informationen darstellt.

Um eine **Hexadezimalzahl** in eine **Dezimalzahl** umzuwandeln, multipliziert man jede Ziffer mit der entsprechenden Potenz der Basis 16 und summiert die Ergebnisse. Die Positionen werden von rechts nach links beginnend bei 0 gezählt:

Hexadezimalzahl in Dezimalzahl umwandeln

16^5	16^4	16^3	16^2	16^1	16^0
1048576	65536	4096	256	16	1

$1 * 4096 + 3 * 256 + 15 * 16 + 8 * 1$

Hexadezimalzahl: **13F8** Dezimalzahl: **5112**

Der entsprechende JavaScript-Befehl:
Dezimalzahl = parseInt(Hexadezimalzahl, 16);

Um eine **Dezimalzahl** in eine **Hexadezimalzahl** umzuwandeln, teilst du die Dezimalzahl wiederholt durch 16 und notiere den Rest. Diese Reste schreibst du dann in umgekehrter Reihenfolge auf (Rote Pfeile im nachfolgenden Bild). Achte darauf, dass die Reste 10 bis 15 in die Hexadezimalziffern A bis F umgewandelt werden (Position 3 im nachfolgenden Bild):

Dezimalzahl in Hexadezimalzahl umwandeln

Dezimalzahl: **5112** Hexadezimalzahl: **13F8**

1 $5112 : 16 = 316 \text{ Rest } 8$	2 $319 : 16 = 19 \text{ Rest } 15 \text{ (F)}$
3 $19 : 16 = 1 \text{ Rest } 3$	4 $19 : 16 = 0 \text{ Rest } 1$

Der entsprechende JavaScript-Befehl:
`Hexadezimalzahl = Dezimalzahl.toString(16);`

Um eine **Hexadezimalzahl** in eine **Binärzahl** umzurechnen, ersetzt du jede einzelne Hexadezimalziffer durch ihren entsprechenden 4-Bit-Binärwert.

Hexadezimalzahl in Binärzahl umwandeln

Hexadezimalzahl: **13F8**

0001	0011	1111	1000
-------------	-------------	-------------	-------------

100111111000

Der entsprechende JavaScript-Befehl:
`Dezimalzahl = parseInt(Hexadezimalzahl, 16);`
`Binärzahl = parseInt(Dezimalzahl, 2);`

Ich habe noch nie eine Hexadezimalzahl direkt in eine Dezimalzahl umgerechnet. Ich gehe immer erst den Weg über die Binärzahl. Warum? Das kann ich immer im Kopf umrechnen. Wenn du einen I2C-Bus-Temperatursensor mit einem Mikrocontroller verheiratest, und der ganze Spaß funktioniert nicht gleich, dann ist während des Debugger-Prozesses die Hexadezimal-Binär-Geschichte dein tägliches Brot. Spätestens danach bist du mit diesem Thema auf Flughöhe.

Die bitweise **UND-Verknüpfung** (englisch: bitwise AND, Symbol $\&$) vergleicht zwei Zahlen Bit für Bit und hat folgende Regel: $1 \& 1 = 1$; $1 \& 0 = 0$; $0 \& 1 = 0$; $0 \& 0 = 0$.

Die bitweise **ODER-Verknüpfung** (englisch: bitwise OR, Symbol $|$) vergleicht zwei Zahlen Bit für Bit und hat folgende Regel: $1 | 1 = 1$; $1 | 0 = 1$; $0 | 1 = 1$; $0 | 0 = 0$.

Die bitweise **Exklusiv ODER-Verknüpfung** (englisch: bitwise XOR, Symbol $\wedge$) vergleicht zwei Zahlen Bit für Bit und hat folgende Regel: $1 \wedge 1 = 0$; $1 \wedge 0 = 1$; $0 \wedge 1 = 1$; $0 \wedge 0 = 0$.

Binäre UND, ODER, exklusiv ODER verknüpfung

1101 0011 & **0000 0011** = **11**

Der entsprechende JavaScript-Befehl für binär UND: `Resultat = a & b / 3 = 211 & 3`

0011 or **1100** = **1111**

Der entsprechende JavaScript-Befehl für binär ODER: `Resultat = a | b / 15 = 3 | 12`

0011 xor **0011** = **0000**

Der entsprechende JavaScript-Befehl für binär xODER: `Resultat = a ^ b / 0 = 3 ^ 3`

Ein bestimmtes Bit in einer **Binärzahl setzen oder zurücksetzen**. Die Eingabe Bitposition (oder Maske) gibt die Nummer des Bits an. Beachte, dass die Nummerierung mit dem rechten Bit mit der Nummer 0 beginnt. Anwendung: Statt für jeden Status (z.B. eingeschaltet/ausgeschaltet) eine eigene Variable zu verwenden, können mehrere Status-Flags in einem einzigen Byte gespeichert werden. Jedes Bit im Byte (8 Bits) wird dann zu einem separaten Schalter. Das Setzen oder Löschen einzelner Bits ermöglicht es, die Datenverarbeitung zu beschleunigen und den Speicherverbrauch zu minimieren, was besonders in der Programmierung von Mikrocontrollern und eingebetteten Systemen wichtig ist.

Bit setzen oder löschen, SetBit or ClearBit

SetBit JavaScript:
`function Set_Bit(num, bit){
 return num | 1<<bit;
}`

1

0010

num

|

0001

1<<bit

return

0011

ODER-Bit Regel Maske

ClearBit JavaScript:
`function Clear_Bit(num, bit){
 return num & ~(1<<bit);
}`

2

0011

num

&

0001

~(1<<bit)

return

0010

UND-Bit Regel Maske

Bitweises Schieben ist eine Operation in der Computerprogrammierung, bei der die Bits eines binären Zahlenwertes nach links oder rechts verschoben werden. Dabei wird der Wert einer Zahl durch das Hinzufügen oder Entfernen von Nullen an den Enden verändert. Diese Operation wird mit den Operatoren `<<` (Left Shift) und `>>` (Right Shift) durchgeführt. Links Schieben ist eine Multiplikation mit 2. Rechts Schieben ist eine Division durch 2. Für die Mikrocontroller-Programmierung bedeutet das, eine schnelle Multiplikation und Division mit Zweierpotenzen, was oft effizienter ist als die Verwendung herkömmlicher arithmetischer Operatoren.

Bitweise schieben, Shift Right, Shift Left

<p>Shift Left JavaScript: <code>function Shift_Left(num, byte){ return num << byte; }</code></p> <p style="text-align: right; color: red; font-weight: bold; border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: inline-block; line-height: 20px;">1</p>	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">0011</div> num	<<	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">2</div> Byte <small>Maske</small>	<<	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">1100</div> return
<p>Shift_Right JavaScript: <code>function Shift_Right(num, byte){ return num >> byte; }</code></p> <p style="text-align: right; color: red; font-weight: bold; border: 1px solid red; border-radius: 50%; width: 20px; height: 20px; display: inline-block; line-height: 20px;">2</p>	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">0111</div> num	>>	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">3</div> Byte <small>Maske</small>	>>	<div style="border: 1px solid white; padding: 5px; display: inline-block; font-size: 24px; font-weight: bold;">0000</div> return