**Seminar 1**

## Darstellung von ganzen Zahlen

Ein Mikroprozessorsystem verarbeitet immer **Bitmuster in Einheiten zu 8, 16, 32 oder mehr Bit. Erst durch die Art der Verarbeitung wird diesem Bitmuster eine bestimmte Bedeutung zugewiesen.**

Wenn einen arithmetischen Maschinenbefehl auf ein Bitmuster angewendet wird, dann wird es als Zahl interpretiert.

### Basis 10

Um die Darstellung der ganzen Zahlen zu verstehen, betrachten wir zunächst das uns geläufige Dezimalsystem, in dem **zehn verschiedene Ziffern mit Potenzen der Zahl 10 gewichtet werden**. Eine Dezimalzahl mit **n Ziffern** hat den Wert:

$$Z = \sum_{i=0}^{n-1} a_i * 10^i$$

Z.B. 123 = $1 * 10^2 + 2 * 10^1 + 3 * 10^0$

### Basis 2 (Binärsystem)

In der Mikroprozessortechnik haben die kleinsten Speichereinheiten, die Bits, nur zwei Zustände. Man hat daher **nur die Ziffern 0 und 1** zur Verfügung und stellt die Zahlen im **Binärsystem** dar. Der Wert einer vorzeichenlosen **Binärzahl** (**unsigned binary numbers**) ist:

$$Z = \sum_{i=0}^{n-1} a_i * 2^i$$

Z.B. 11100101b = $1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

Um Binärzahlen von Dezimal- u.a. Zahlen zu unterscheiden, wird an die Ziffernfolge der Buchstabe **'b' angehängt** (1101b) oder **die Zahlenbasis als tiefgestellter Index** ($1101_2$).

### Basis 16 (Hexadezimalsystem)

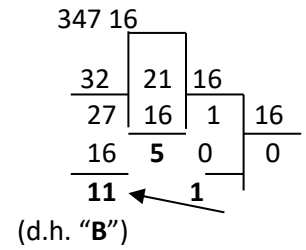
Da nur 10 Zahlenzeichen zur Verfügung stehen, verwendet man die ersten sechs Buchstaben des Alphabets für die Zahlen 10 bis 15.

Die Standardeinheit der Informationsgröße ist ein Byte, das sind 8 Bit. Diese Einheit lässt sich mit dem 16er-System viel besser handhaben als mit dem Dezimalsystem, denn 256 ist gerade $16^2$. Somit entsprechen in **einer Hexadezimalzahl** immer **genau zwei Ziffern einem Byte**.

### Umwandeln von Dezimalzahlen in eine andere Zahlenbasis

1) Dividieren Sie die Dezimalzahl durch die Ziel-Zahlenbasis und merken Sie sich das Ergebnis und den Restwert.
2) Dividieren Sie weiter das vorige Ergebnis durch die Ziel-Zahlenbasis bis Sie ein Ergebnis von 0 erhalten.
3) Der Wert in der Ziel-Zahlenbasis ergibt sich aus den Restwerten in umgekehrte Reihenfolge.

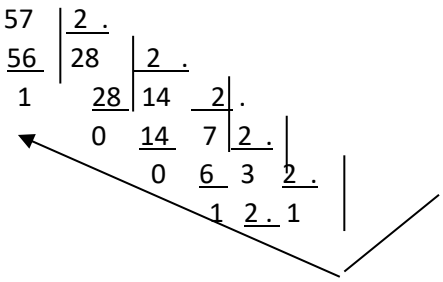
Z.B. Wandle die Zahl 347 in die Basis 16 um:




(d.h. "**B**")

**347 = 15B$_{(h)}$**

Wandle die Zahl 57 in die Basis 2 um:




**57$_{(10)}$ = 111001$_{(2)}$**

Andere Beispiele:

$2 = 2_{(10)} = 10_{(2)}$ ;

$62_{(10)} = 111110_{(2)}$ ;

$1995_{(10)} = 11111001011_{(2)}$ ;

$1024_{(10)} = 10000000000_{(2)}$ ;

**<u>Umwandeln von Zahlen aus einer beliebigen Zahlenbasis in die Dezimalbasis</u>**

Um eine Zahl aus einer Basis b in die Dezimalbasis umzuwandeln benutzen wir folgende Formel.

Für die Zahl: $\mathbf{Nr(b) = C_n C_{n-1} C_{n-2} \dots C_2 C_1 C_0}$

ist der Wert in der Dezimalbasis:

$$\mathbf{Nr(10) = C_n * b^n + C_{n-1} * b^{n-1} + \dots + C_2 * b^2 + C_1 * b^1 + C_0}$$

Z.B.

- Wandle die Zahl $3A8_{(h)}$ in die Dezimalbasis um:
  $N = 3*16^2 + 10*16^1 + 8 = 3*256 + 160 + 8 = 936_{(10)}$
- Wandle die Zahl $86C_{(h)}$ in die Dezimalbasis um:
  $86C_{(16)} = 8 * 16^2 + 6 * 16 + 12 = 2156_{(10)}$
- Wandle die Zahl $1101101_{(2)}$ in die Dezimalbasis um:
  $1101101_{(2)} = 1*2^6+1*2^5+0*2^4+1*2^3+1*2^2+0*2+1=109_{(10)}$

**Umwandeln von Binär- in Hexadezimalzahlen und umgekehrt**

Jeweils 4 Binärstellen entsprechen einer Hexadezimalstelle, denn $16 = 2^4$. Daher lassen sich diese Systeme auch ohne Umweg direkt und stellenweise umwandeln:

- **Vom Binär- ins Hexadezimalsystem:**

Unterteile die Binärzahl **von rechts nach links** in 4er-Päckchen, und wandle jedes Päckchen nach nebenstehender Tabelle in die entsprechende Hexadezimalziffer um.

- **Vom Hexadezimal- ins Binärsystem:**

Wandle die Hexadezimalziffern der Reihe nach in die entsprechenden vierstelligen Binärzahlen um.

| Hexadezimal | Binär | Hexadezimal | Binär |
|---|---|---|---|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | A | 1010 |
| 3 | 0011 | B | 1011 |
| 4 | 0100 | C | 1100 |
| 5 | 0101 | D | 1101 |
| 6 | 0110 | E | 1110 |
| 7 | 0111 | F | 1111 |

Z.B.

- $49A02_{(16)} = 0100\ 1001\ 1010\ 0000\ 0010_{(2)}$
- $10010110101011_{(2)} = 0010\ 0101\ 1010\ 1011_{(2)} = 25AB_{(16)}$
- $347 = (15B)_{16} = (0001\ 0101\ 1011)_2$
- $(9A7D)_{16} = (1001\ 1010\ 0111\ 1101)_2$
- $(1\ 0010\ 1110)_2 = (12E)_{16}$
- $(1\ 1100\ 1110\ 0000\ 1101)_2 = (1CE0D)_{16}$
- $(26B)_{16} = (0010\ 0110\ 1011)_2$

**Bit. Byte.**

Die **kleinste Informationseinheit**, das **Bit**, hat den Wert 1 oder 0. **Erst durch die Art der Verarbeitung wird diesem Bitmuster eine bestimmte Bedeutung zugewiesen.**

Die mögliche Werte können z.B. die Zahlen 0 oder 1, die Wahrheitswerte wahr oder falsch, positiv oder negativ, usw. darstellen.

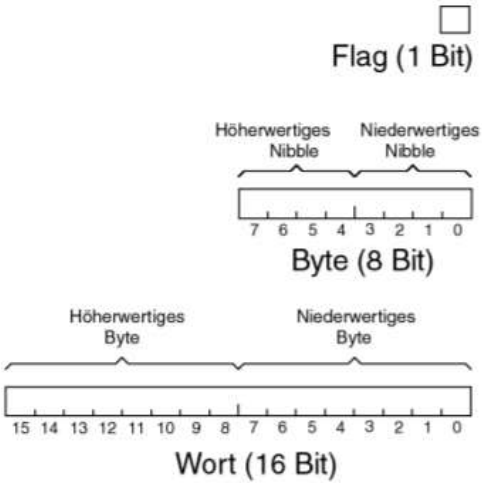
Eine Einheit aus **4 Bit** heißt **Tetrade** oder **Nibble**.

Eine **8 Bit-Einheit** heißt **Byte** .

Ein **Wort (word) ist eine 16 Bit-Einheit.**

Ein **Doppelwort ist eine 32 Bit-Einheit.**

Innerhalb einer Einheit sind die Bits nummeriert. Das niederwertigste Bit, das **Least significant Bit**, abgekürzt das LSB , ist immer Bit 0. Das höchstwertige Bit, das **Most significant Bit** , abgekürzt das MSB , ist bei einem Byte Bit 7, bei einem 16 Bit-Wort Bit 15.




**<u>Repräsentierung der negativen Zahlen</u>**

Ein weiterer Punkt, der beim Arbeiten mit Binär-Zahlen zu beachten ist, ist die Darstellung von negativen Werten. Bei der alltäglichen Benutzung können wir einfach ein Minus vor die Dezimalzahl setzen. Wir können dies jedoch nicht mit Binärzahlen machen.

Eine Lösung ist, ein Bit für die Darstellung des Vorzeichens zu benutzen, das **Vorzeichenbit (sign bit)** genannt wird. Das Vorzeichenbit hat den Wert **1 für negative Zahlen** und den Wert **0 für positive Zahlen**.

Das Vorzeichenbit ist das **Most Significant Bit (das erste Bit von links).**

Es wurden unterschiedliche Methoden vorgeschlagen um eine negative Zahl darzustellen:

1) <u>Direkter Kod</u>

Am einfachsten wäre den positiven Wert darzustellen und das Most significant Bit auf 0 für positive oder auf 1 für negative Zahlen zu setzen

z.B. repräsentiere 5 und -5 auf einem Byte:

$5 = 00000101_{(2)}$ , $-5 = 10000101_{(2)}$

**Problem:** $5 + (-5) = 10001010_{(2)} \neq 0$

2) <u>Einerkomplement</u>

Das Einerkomplement sagt, dass alle positiven Zahlen so bleiben wie sie sind und alle negativen invertiert werden, d.h. wenn man alle Ziffern der Darstellung des positiven Wertes negiert (aus der Ziffer 0 wird 1, aus der 1 wird 0).

Z.B. $7 = 0111_{(2)}$ , $-7 = 1000_{(2)}$

**Problem**: $7 - 7 = 1111_{(2)} \neq 0$

**3) <u>Zweierkomplement</u>**

Zweierkomplement wird für die Darstellung der negativen Zahlen in der Praxis benutzt!

**Die Vorzeichenumkehr einer Zahl im Zweierkomplement wird bewirkt durch Invertieren aller Bits und anschließendes Inkrementieren.**

Andere äquivalente Komplementierungsregeln:

- Von rechts angefangen, alle Nullen und die erste Eins abschreiben und alle nachfolgenden Stellen invertieren.
- Subtraktion von der Wertebereichsgrenz: um das Komplement für eine binäre Zahl mit n Ziffern zu berechnen, subtrahiert man binär die Zahl aus $10\ldots0_{(b)}$, wobei diese Zahl n 0-Ziffern hat
- Subtraktion von der Wertebereichsgrenz: um das Komplement für eine hexadezimale Zahl mit n Ziffern zu berechnen, subtrahiert man hexadezimal die Zahl aus $10\ldots0_{(h)}$, wobei diese Zahl n 0-Ziffern hat

Z.B.

| Bitmuster | Wert dezimal |
|---|---|
| 11111101 | -3 |
| 11111110 | -2 |
| 11111111 | -1 |
| 00000001 | 1 |
| 00000010 | 2 |
| 00000011 | 3 |

-1 + 1 führt zu dem richtigen Ergebnis 0.

Vorteile der Zweierkomplement:

- Die Additionsoperation wird gleich ausgeführt, egal ob es um negative oder positive Zahlen geht. Die Addition ist eine normale bitweise Addition, wo der Stellenüberlauf ignoriert wird. Das Ergebnis wird mit gleicher Stellenanzahl wie Minuend und Subtrahend interpretiert.
- Die Subtraktion lässt sich auf eine Negation mit anschließender Addition zurückführen.

Z.B. Berechne das Zweierkomplement für die Zahl $(18)_{10}$ repräsentiert auf einem Byte:

| | |
|---|---|
| Ursprünglich: | 00010010 |
| Nach Invertierung aller Bits: | 11101101 |
| Addiere 1: | 11101101+ |
| | <u>00000001</u> |
| Komplement: | 11101110 |

Also die Zahl $(18)_{10} = (12)_{16} = (00010010)_2$ hat als Komplement die Zahl $(11101110)_2 = (EE)_{16} = (238)_{10}$ .

Binäre Subtraktion von der Wertebereichsgrenz:

| | |
|---|---|
| | 100000000- |
| Ursprünglich: | <u>00010010</u> |
| Komplement: | 11101110 |

Hexadezimale Subtraktion von der Wertebereichsgrenz:

| | |
|---|---|
| | 100- |
| Ursprünglich iniţială: | 12 |
| Komplement: | EE |

Bsp. Überprüfe mit mehreren Regeln, dass:

- die Zahlen $(9A7D)_{16}$ und $(7583)_{16}$ repräsentiert auf 2 Bytes komplementär sind.
- die Zahlen $(000F095D)_{16}$ und $(FFF0F6A3)_{16}$ repräsentiert auf 4 Bytes komplementär sind.
- die Zahlen $(7F)_{16}$ und $(81)_{16}$ repräsentiert auf 1 Byte komplementär sind.

**Darstellung der ganzen Zahlen:**

Eine Zahl zwischen $\mathbf{-2^{n-1}}$ **und** $\mathbf{2^{n-1} - 1}$ wird auf **n Bits** folgendermaßen repräsentiert:

- falls die Zahl positiv ist, dann wird sie durch den binären Wert dargestellt
- falls die Zahl negativ ist, dann wird sie durch den Zweierkomplement der binären Wert dargestellt

**Bem.** Die Zahl $\mathbf{-2^{n-1}}$ kann nicht auf n-1 Bits dargestellt werden. $\mathbf{-2^{n-1}}$ wird **auf n Bits als 10…0** dargestellt, wobei die Most Significant Bit 1 ist, da die Zahl negativ ist. Die Zahl 10…0 hat sich selbst als Komplement, also der Wert ist genau $\mathbf{-2^{n-1}}$.

Dieselbe Zahl 10…0 interpretiert als natürliche Zahl ist die Zahl $\mathbf{2^{n-1}}$.

**Darstellungsbereiche**

Da in der Mikroprozessortechnik immer die Bitstellenzahl begrenzt ist, ist auch der Zahlenbereich begrenzt. Zahlen außerhalb dieses Bereichs sind nicht darstellbar und eine Operation, deren Ergebnis über eine der Grenzen hinausführt, ergibt ein falsches Ergebnis. Diese **Bereichsüberschreitung** wird vom Mikroprozessor mit dem **Übertragsflag** (**Carryflag**) angezeigt.

| Anzahl Bytes | Nicht signierte Interpretation (vorzeichenlosen, unsigned representation) | Signierte Interpretation (vorzeichenbehafteten, signed representation) |
|---|---|---|
| 1 | $[0, 2^8-1] = [\ 0\ ,\ 255\ ]$ | $[-2^7\ ,\ 2^7-1] = [-128\ ,\ 127]$ |
| 2 | $[0, 2^{16}-1] = [\ 0\ ,\ 65535\ ]$ | $[-2^{15}\ ,\ 2^{15}-1] = [-32\ 768\ ,\ 32\ 767]$ |
| 4 | $[0, 2^{32}-1] = [\ 0\ ,\ 4\ 294\ 967\ 295\ ]$ | $[-2^{31}\ ,\ 2^{31}-1] = [-2\ 147\ 483\ 648\ ,\ 2\ 147\ 483\ 647]$ |
| 8 | $[0, 2^{64}-1] = [\ 0\ ,\ 18\ 446\ 824\ 753\ 389\ 551\ 615\ ]$ | $[-2^{63}\ ,\ 2^{63}-1] = [-9\ 223\ 412\ 376\ 694\ 775\ 808\ ,\ 9\ 223\ 412\ 376\ 694\ 775\ 807]$ |

Frage: Wie wird eine Zahl auf 7 Bits auf 8 Bits dargestellt?

Antwort: es hängt von der Interpretation ab:

- in der vorzeichenlosen Interpretation werden die restlichen höherwertige Bits (high bits) mit Nullwerte ausgefüllt.
- in der vorzeichenbehafteten Interpretation werden die restlichen höherwertige Bits (high bits) mit dem Vorzeichenbit (sign bit) ausgefüllt.

Z.B. $(10011)_2$ wird auf einem Byte folgendermaßen dargestellt:

- $(00010011)_2$ in der vorzeichenlosen Darstellung
- $(11110011)_2$ in der vorzeichenbehafteten Darstellung

*Es gibt 10 Gruppen von Menschen: diejenigen, die das Binärsystem verstehen, und die anderen.*