

# GF Informatik: Zahlensysteme

Andreas Schärer

1M

2021-2022



# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>1</b>
<b>2</b>	<b>Zahlensysteme</b>	<b>2</b>
<b>3</b>	<b>Binärsystem</b>	<b>3</b>
3.1	Grundlagen . . . . .	3
3.2	Addition . . . . .	11
3.3	Computer-Architekturen . . . . .	13
3.4	Negative Zahlen & Subtraktion . . . . .	14
3.5	Multiplikation . . . . .	18
<b>4</b>	<b>**Optional: Hexadezimalsystem</b>	<b>20</b>

# 1 Einführung

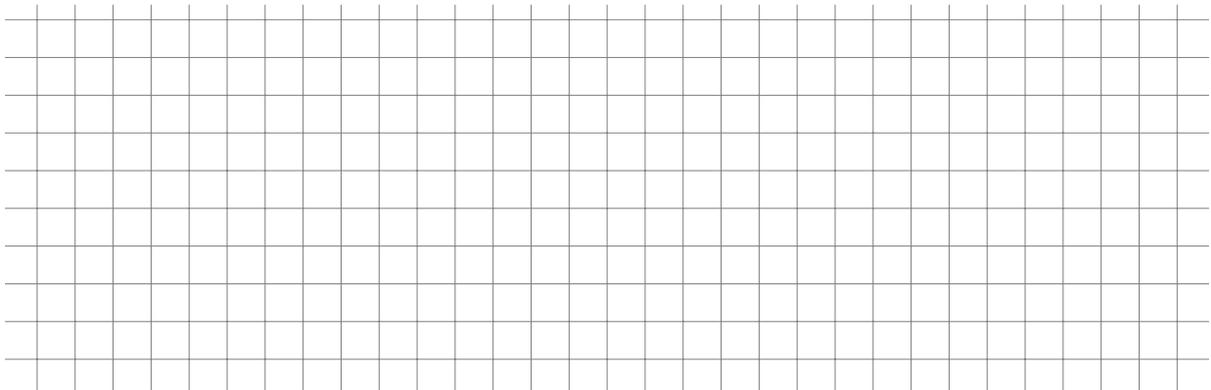
Wir sind es uns gewohnt, im Dezimalsystem, also dem 10er-System, zu arbeiten. Ein Computer hingegen arbeitet im Dualsystem (Binärsystem), also dem Zahlensystem, welches nur aus Nullen und Einsen besteht. Zum Beispiel hat die Dezimalzahl 37 im Binärsystem die Form 100101.

Doch warum rechnet ein Computer nur mit 0 und 1? Ein Computer besteht aus vielen elektronischen Bauteilen, in denen entweder Strom fließen kann oder eben nicht. Die 1 steht dabei für 'es fließt Strom' und die 0 für 'es fließt kein Strom'. Diese beiden Zustände können mit einem **Bit** dargestellt werden. Ein Bit ist die kleinste mögliche Informationseinheit. Sie hat zwei Möglichkeiten: 0 oder 1. Die Binärzahl 100101 besteht also aus 6 Bits.

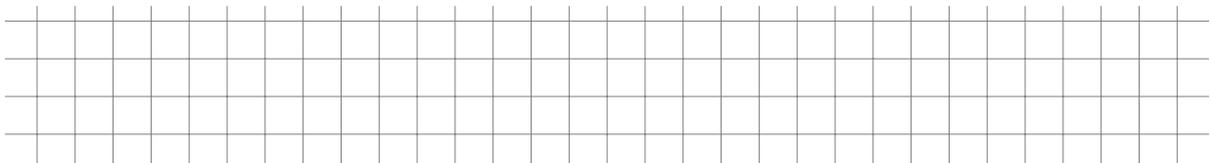
In diesem Dossier geht es darum, wie man Zahlen in Zahlensystemen darstellen und interpretieren kann. Betrachten wir die Zahl 100. Wahrscheinlich denkst du sofort 'Hundert'! Dies ist aber nur der Fall, wenn wir diese Zahl im uns bekanntesten Zahlensystem, dem Dezimalsystem, betrachten. Betrachtet man diese Zahl hingegen im Binärsystem, so hat es den Wert 4 im Dezimalsystem. Damit man eine Zahl sinnvoll interpretieren kann, muss man also immer wissen, in welchem Zahlensystem sie dargestellt wird.

Wir werden uns hier hauptsächlich mit dem Dualsystem beschäftigen und schauen, wie dort Grundoperationen wie das Addieren und Subtrahieren funktionieren.

**Beispiel: Mit den Fingern zählen.** Wie weit kann man mit einer Hand zählen? Mache für die ersten paar Beispiele eine Skizze?



Wie weit kann man mit zwei Händen zählen?



## 2 Zahlensysteme

### Definition 2.1

Ein **Zahlensystem** ist ein System, mit dem Zahlen dargestellt werden. Es wird durch seine **Basis** und seine **Nennwerte** festgelegt.

Beispiele für Zahlensysteme sind das uns sehr vertraute Dezimalsystem (Basis 10), das Binärsystem (Basis 2) oder das Hexadezimalsystem (Basis 16).

Im **Dezimalsystem** (auch **Zehnersystem**) ist die Basis 10 und die Nennwerte sind 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9. Die Zahl  $1903_{10}$  ist dann wie folgt zu interpretieren:

$$1903_{10} = 1 \times 10^3 + 9 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 \quad (1)$$

Mit der *kleinen Zahl unten rechts* (10) deuten wir an, dass die Zahl im Dezimalsystem zu betrachten ist. Lässt man diese Zahl weg, schreibt man also z.B. 576, so bedeutet dies (meistens), dass die Zahl im Dezimalsystem steht.

### Aufgabe 2.1

- Teile die Zahl  $41086_{10}$  auf wie im Beispiel oben. Was sind ihre Nennwerte? Was ist die Basis?
- Was ist die Basis im Dualsystem? Was sind die Nennwerte?



## 3 Binärsystem

### 3.1 Grundlagen

*“I told my friend 10 jokes about the binary system - He didn't get either of them!”*

#### Definition 3.1

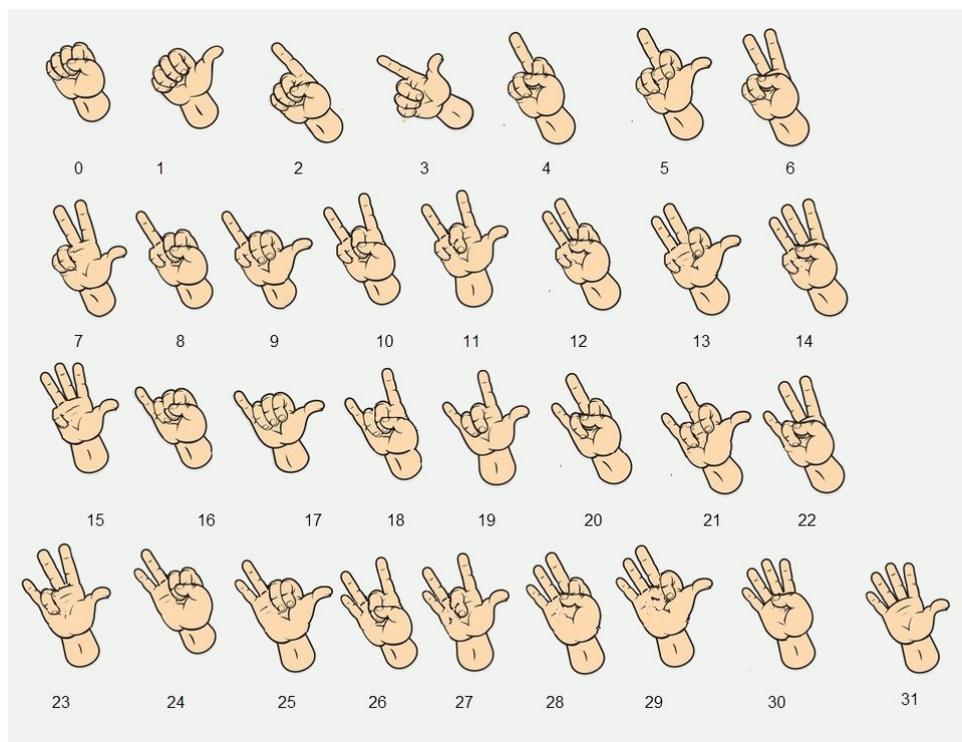
Die kleinste Informationseinheit ist das **Bit**, es hat zwei Möglichkeiten: es kann entweder 0 oder 1 sein. In der Welt der Elektrotechnik hat diese eine besondere Relevanz, da diese den beiden Zuständen ‘es fließt kein Strom (0)’ oder ‘es fließt Strom (1)’ entsprechen.

Deshalb ist da das **Binärsystem** (auch **Dualsystem** oder **Zweiersystem**) wichtig: Die Basis ist 2 und die Nennwerte sind 0 und 1. Eine Binärzahl besteht also aus mehreren Bits.

Das **Umrechnen einer Binärzahl in eine Dezimalzahl** geht ganz einfach. Für die Zahl  $100101_2$  geht man wie folgt vor:

$$\begin{aligned} 100101_2 &= 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32_{10} + 4_{10} + 1_{10} \\ &= 37_{10} \end{aligned} \tag{2}$$

Im Bild unten siehst du, wie man mit den Fingern einer Hand binär bis auf 31 zählen kann.





**Aufgabe 3.2**

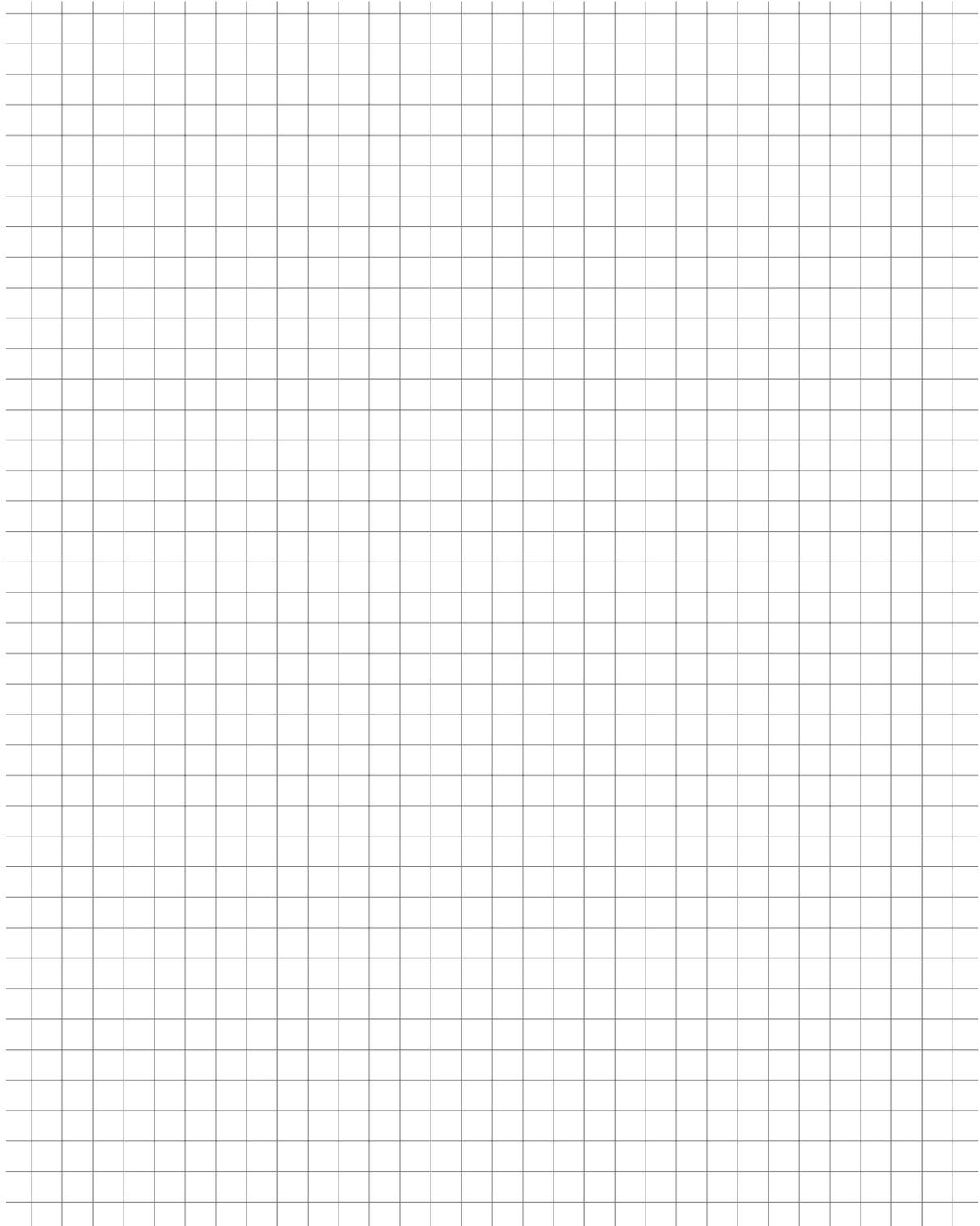
Wandle die Binärzahlen ins Dezimalsystem um. Gehe dabei *rechnerisch* vor. Jeder Rechenschritt muss dabei klar ersichtlich sein.

a)  $111_2$

b)  $1000011_2$

c)  $1101010_2$

d)  $100010001000_2$



### Aufgabe 3.3

**Code: Binärzahl zu Dezimalzahl.** Schreibe in Python eine Funktion `binary_to_decimal(b)`, welche eine Binärzahl `b` in die zugehörige Dezimalzahl umwandelt und zurückgibt. Überprüfe deinen Code, indem du diesen auf die Beispiele aus der letzten Aufgabe anwendest.

Dazu einige Tipps:

- Schreibe die Binärzahl als String, z.B. `b = '100101'`.
- Ein String kann dann behandelt werden wie ein Array. Mit `b[2]` kann man den dritten Buchstaben des Strings `b` auslesen. Auch kann man genau gleich mit einer for-Schleife (`for x in b: ...`) durch alle Buchstaben des Strings durchgehen.

Füge hier einen Screenshot deines Codes ein:



Wir wissen nun, wie man Binärzahlen in Dezimalzahlen umwandelt. Ziel dieser Aufgabe ist herauszufinden, wie der umgekehrte Schritt funktioniert: Das **Umwandeln von Dezimalzahlen in Binärzahlen**.

Dazu gibt es einen *Algorithmus*, welcher wie folgt funktioniert: Starte mit der gegebenen Dezimalzahl und dividiere sie durch 2 und notiere das Resultat sowie den Rest. Wiederhole nun diesen Schritt, allerdings startest du nun mit dem Resultat des letzten Schritts. Dies wiederholst du solange, bis als Resultat 0 herauskommt. Die Binärzahl ist dann gegeben durch die Reste der Divisionen und zwar in umgekehrter Reihenfolge. Dieser Algorithmus wird der **Restwert-Algorithmus** genannt.

Schauen wir uns diesen Algorithmus für die Zahl  $42_{10}$  an:

Schritt 1:	42	/	2	=	21	Rest:	0
Schritt 2:	21	/	2	=	10	Rest:	1
Schritt 3:	10	/	2	=	5	Rest:	0
Schritt 4:	5	/	2	=	2	Rest:	1
Schritt 5:	2	/	2	=	1	Rest:	0
Schritt 6:	1	/	2	=	0	Rest:	1

Das Resultat ist also

$$42_{10} = 101010_2$$

Damit man diese Rechnung auch etwas kürzer - aber doch übersichtlich - darstellen kann, bietet sich folgende Darstellung an:

<b>42</b>	
21	0
10	1
5	0
2	1
1	0
0	1

**Aufgabe 3.5**

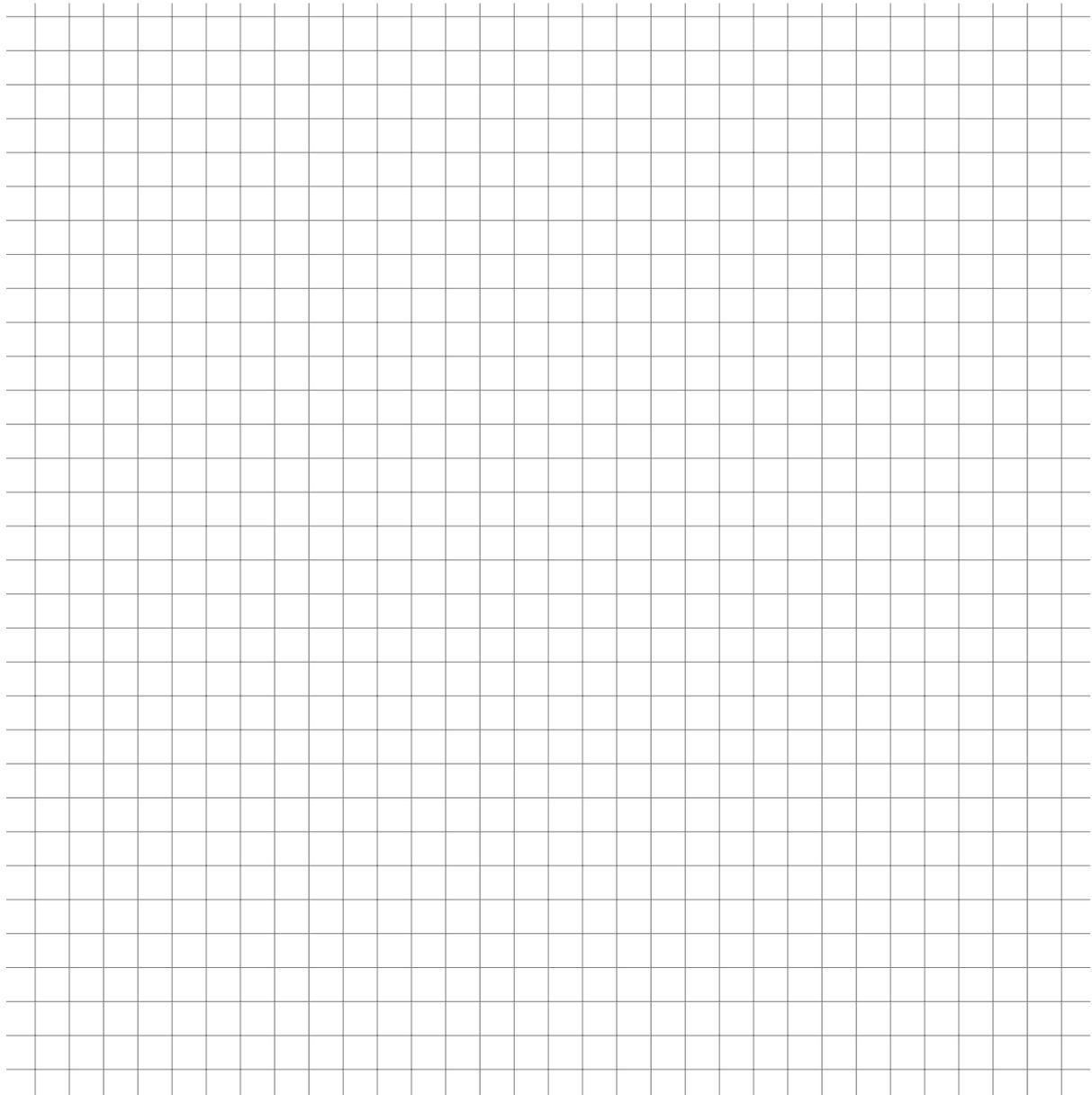
Wandle folgende Dezimalzahlen mit Hilfe des Restwert-Algorithmus in Binärzahlen um. Achte auf eine saubere und klare Darstellung.

a)  $13_{10}$

b)  $19_{10}$

c)  $217_{10}$

d)  $56\,379_{10}$



**Aufgabe 3.6**

**Code: Dezimalzahl zu Binärzahl mit Restwert-Algorithmus.** Schreibe eine Funktion `decimal_to_binary(d)`, welche eine Dezimalzahl `d` in die zugehörige Binärzahl umrechnet und zurückgibt. Implementiere dazu den Restwert-Algorithmus. Überprüfe deinen Code, indem du diesen auf die Beispiele aus der letzten Aufgabe anwendest.

Füge hier einen Screenshot deines Codes ein:

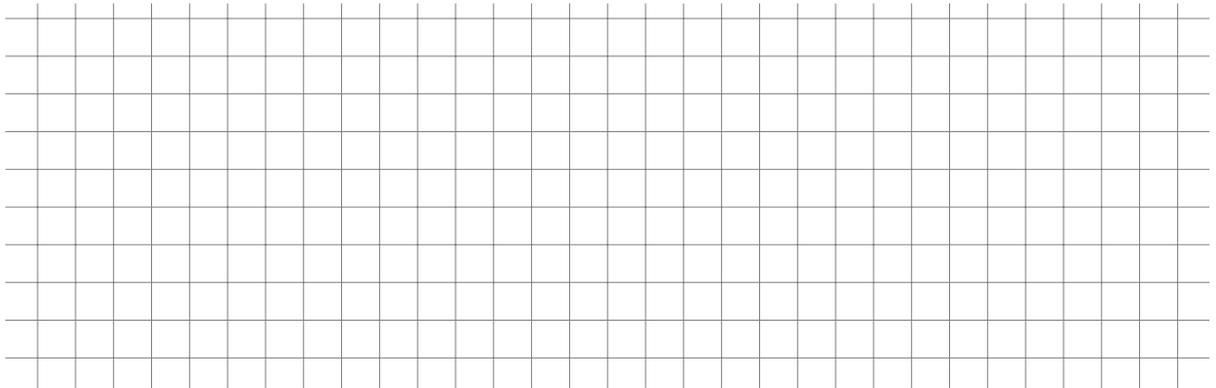
## 3.2 Addition

### Aufgabe 3.7

Das **schriftliche Addieren von zwei binären Zahlen** funktioniert nach dem gleichen Prinzip wie das schriftliche Addieren von Dezimalzahlen, welches man in der Primarschule lernt.

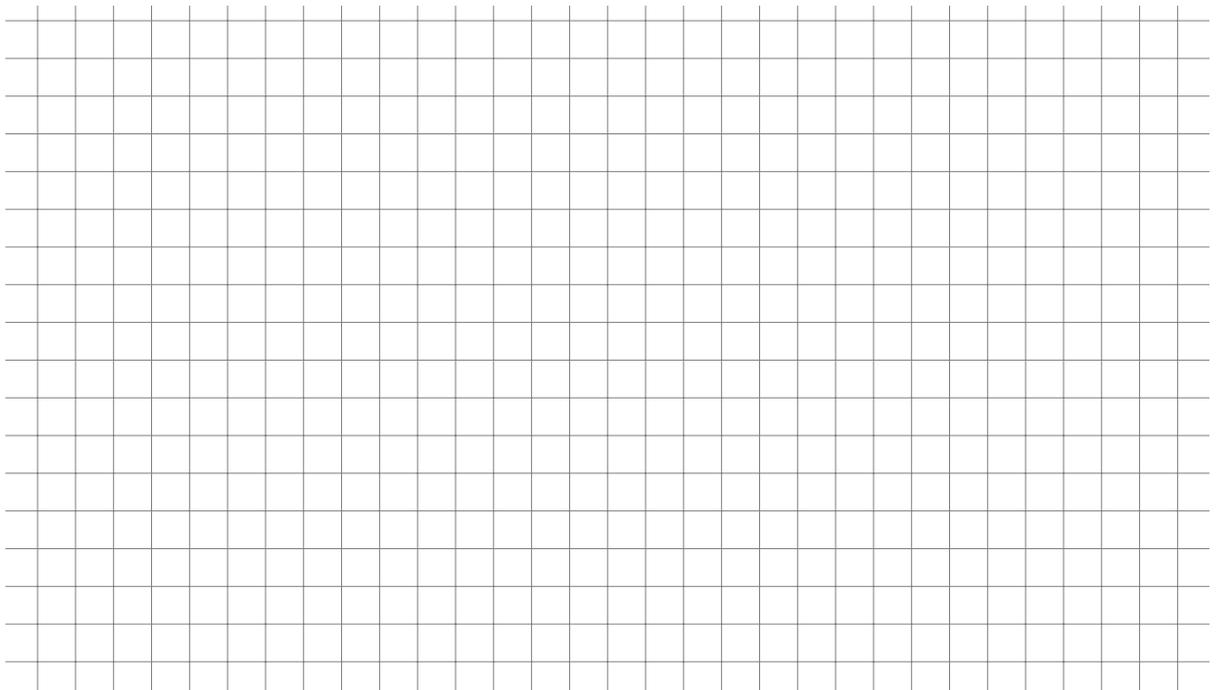
- a) Führe zum Aufwärmen die folgende Addition zweier Dezimalzahlen schriftlich aus:

$$50\,974_{10} + 81\,963_{10}$$



- b) Führe nun folgende Additionen schriftlich aus:

- $1100\,1100_2 + 0011\,0011_2$
- $1100\,1011_2 + 1010\,1110_2$
- $1101\,0010_2 + 1100\,0101_2$
- $1110\,1111_2 + 1011\,0101_2$





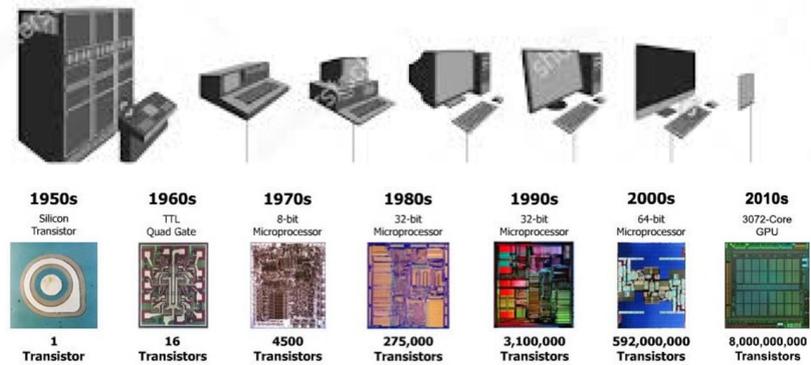
### Aufgabe 3.8

**Code: Addition von Binärzahlen.** Schreibe nun eine Funktion `binary_add(a,b)` welches zwei Binärzahlen `a` und `b` (als Strings gegeben) addiert und zurückgibt. Dabei soll vorgegangen werden wie gerade eben gelernt. Es ist nicht erlaubt, die Binärzahlen in Dezimalzahlen umzurechnen und dann zu addieren.

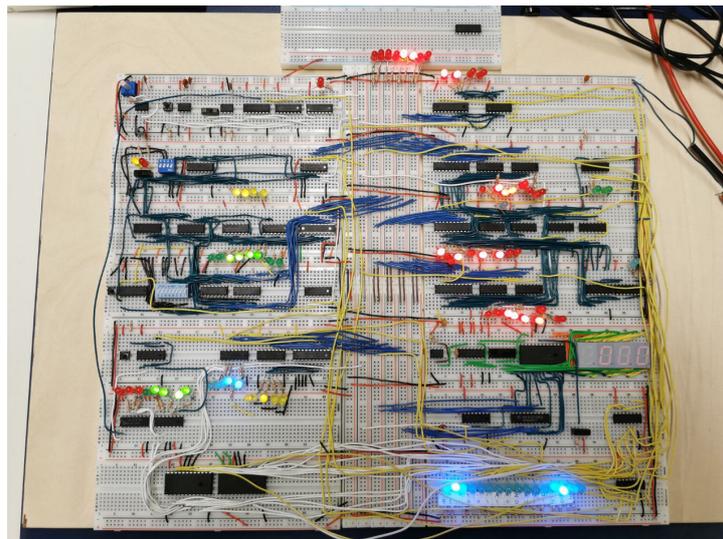
Füge hier einen Screenshot deines Codes ein:

### 3.3 Computer-Architekturen

Moderne Computer wie dein Laptop haben eine **64-Bit Architektur**. Das heisst, dass der Prozessor mit 64-Bit Zahlen rechnet, also Zahlen, die aus 64 Nullen und Einsen bestehen. Dies war aber noch nicht immer der Fall, wie die Grafik unten zeigt:



Das folgende Bild zeigt eine 8-Bit CPU und wurde von TALIT-Schülern gebaut:



### 3.4 Negative Zahlen & Subtraktion

In der Mathematik ist die Subtraktion keine eigene Operation, denn sie ist durch die Addition definiert:  $a - b = a + (-b)$ , oder: 'Subtraktion' = 'Addition der Gegenzahl'. Damit man eine Binärzahl von einer anderen subtrahieren kann, muss man deshalb zuerst festlegen, wie man **negative Zahlen** ausdrückt.

Man könnte sich zum Beispiel darauf einigen, dass das Bit ganz links über das Vorzeichen entscheidet und der Absolutwert der Zahl durch die restlichen Bits festgelegt ist. Dann wäre  $0101_2 = +5_{10}$  und  $1101_2 = -5_{10}$ . Dies birgt aber ein Problem: Addiert man eine Zahl mit seiner Gegenzahl, so möchte man 0 erhalten. Dies ist mit dieser Definition aber nicht der Fall:  $0101_2 + 1101_2 = 10010_2 \neq 0$ . Deshalb verwerfen wir diesen Ansatz, um negative Zahlen darzustellen.

Stattdessen wird eine negative Zahl durch ihr **2er-Komplement** ausgedrückt.

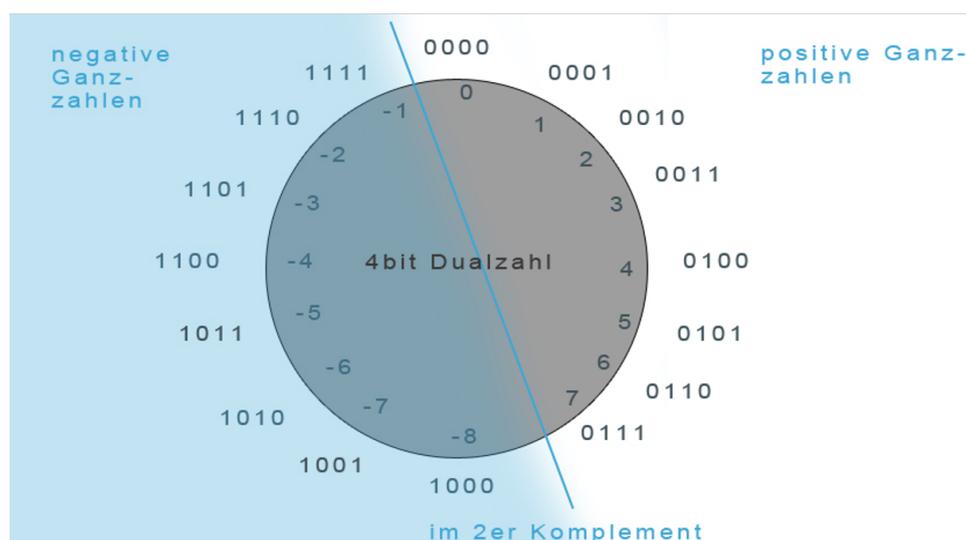
Betrachten wir eine 4-Bit CPU. Jeder Zahl stehen also 4 Bits zur Verfügung - egal ob sie positiv oder negativ ist. Die Dezimalzahl 5 ist offensichtlich  $0101_2 = +5_{10}$ . Die Gegenzahl  $-5$  soll nun durch diejenige 4-Bit Zahl ausgedrückt werden, die in der Summe mit  $0101$  genau  $0000$  ergibt:

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{1} \phantom{0} \phantom{1} \\ + \phantom{0} \phantom{?} \phantom{?} \phantom{?} \phantom{?} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

Wie du dich unschwer selbst davon überzeugen kannst, muss man für die vier Fragezeichen die Zahlen  $1011$  einsetzen, die Antwort ist also:  $-5_{10} = 1011_2$ . Machen wir die Kontrolle:

$$0101_2 + 1011_2 = 10000_2$$

Beachte, dass wir im fünften Bit eine Eins erhalten - dieses Bit ist aber irrelevant, da wir nur in 4 Bits arbeiten. Eine 4-Bit CPU kann dieses 5. Bit gar nicht handeln und es fällt einfach weg! Die folgende Grafik zeigt alle ganzen Zahlen, die eine 4-Bit CPU darstellen kann:



Man sagt, dass  $1011_2$  das **2er-Komplement** von  $0101_2$  in 4-Bit ist.

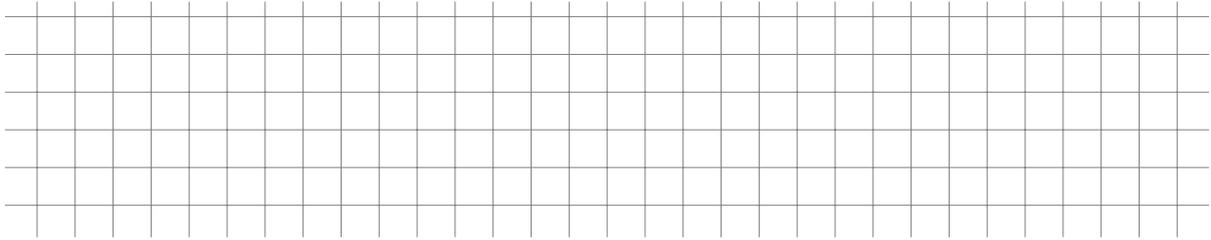
Das **2er-Komplement** kann man nach folgendem Rezept bilden:

1. Die Zahl wird in der gewünschten Anzahl Bits ausgedrückt.
2. Dann werden alle Bits invertiert ( $0 \rightarrow 1, 1 \rightarrow 0$ ) ...
3. und am Schluss 1 dazu addiert.

Das Bit ganz links entscheidet hier also darüber, ob die Zahl positiv oder negativ ist.

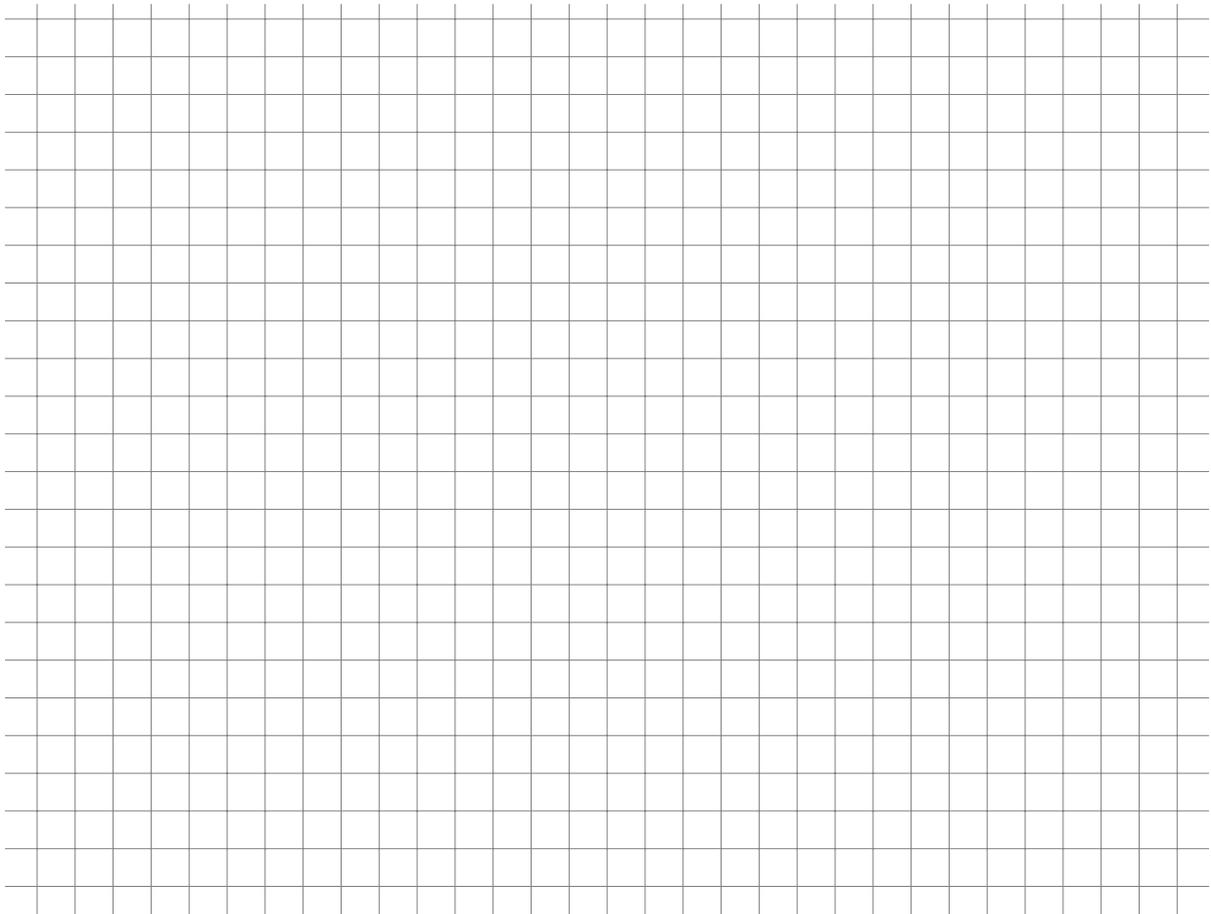
**Aufgabe 3.9**

- a) Bestimme die Gegenzahl von  $101101_2$  in 8-Bit.
- b) Welche negative Dezimalzahl wird durch die Zahl  $10101011_2$  ausgedrückt?

**Aufgabe 3.10**

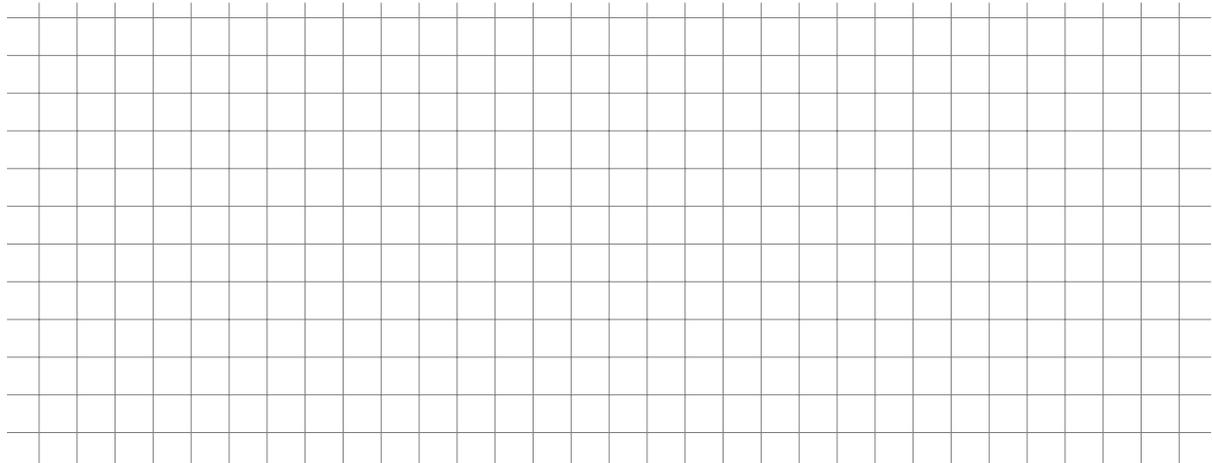
Führe die folgenden Subtraktionen aus. Bestimme dazu vom Subtrahend das 2er-Komplement und addiere dann die beiden Zahlen. Die einzelnen Zahlen sind als positive Zahlen zu interpretieren. Um das 2er-Komplement zu bilden, müssen sie also um ein Bit erweitert werden.

- a)  $10000000_2 - 00011111_2$
- b)  $10101011_2 - 01101011_2$
- c)  $11110010_2 - 10001111_2$



**Aufgabe 3.11**

Mit 4-Bits sollen ganze Zahlen (also positiv und negativ) ausgedrückt werden. Bestimme die grösst- und kleinstmögliche Zahl sowohl als Binär- wie auch als Dezimalzahl.

A large grid consisting of 20 columns and 15 rows of small squares, intended for the student to write their answer to Aufgabe 3.11.**Aufgabe 3.12**

- a) **Code: 2er-Komplement.** Schreibe eine Funktion `binary_complement(b)`, der von einer Binärzahl das 2er-Komplement bestimmt. *Tipp:* Verwende deinen Code zur Addition von Binärzahlen, den du bereits geschrieben hast.
- b) **Code: Subtraktion von Binärzahlen.** Schreibe eine Funktion `binary_sub(a,b)`, welche die Binärzahl `b` von der Binärzahl `a` (beide als Strings gegeben) subtrahiert. *Tipp:* Verwende den Code aus der vorherigen Teilaufgabe.

Füge hier einen Screenshot deines Codes ein:

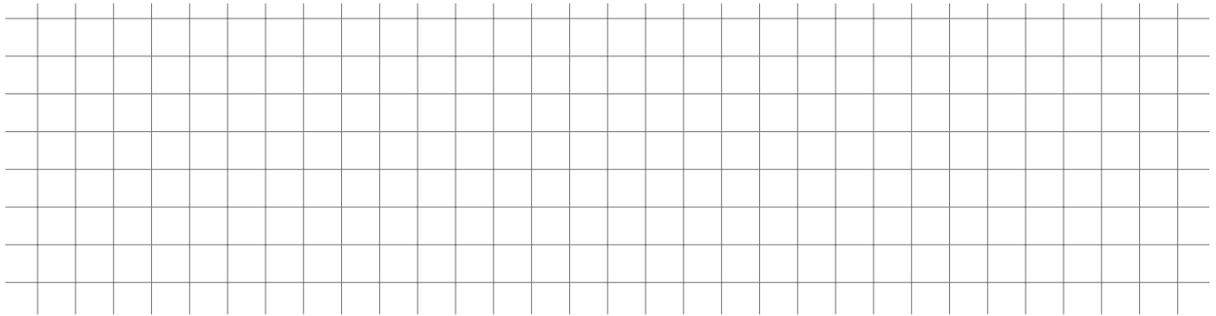
### 3.5 Multiplikation

#### Aufgabe 3.13

Auch die **schriftliche Multiplikation von Binärzahlen** funktioniert analog zur derjenigen von Dezimalzahlen.

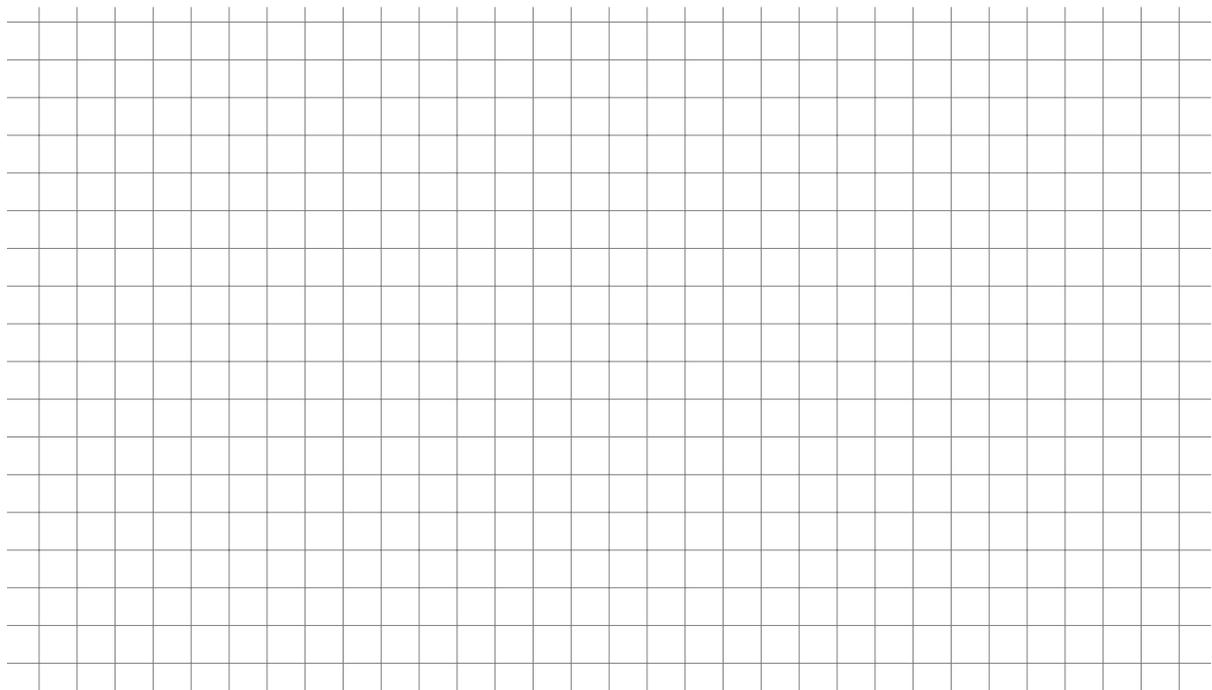
- a) Multipliziere zur Auflockerung zuerst folgende Multiplikation *schriftlich* aus:

$$7905_{10} \cdot 3815_{10}$$



- b) Multipliziere nun die folgenden Binärzahlen schriftlich. Achtung: Besonders aufpassen hier muss man mit den Überträgen am Schluss beim Aufsummieren. Ist die Summe 5, so notiert man eine 1 und bei der *übernächsten* Stelle einen Übertrag von 1, da  $5_2 = 10_2$ .

- $1101_2 \cdot 1111_2$
- $10010010_2 \cdot 01100011_2$
- $11111011_2 \cdot 11101011_2$
- $1100101100111101_2 \cdot 1110011101111010_2$



**Aufgabe 3.14**

**Code: Multiplikation von Binärzahlen.** Schreibe eine Funktion `binary_mult(a,b)`, welche zwei Binärzahlen multipliziert und zurückgibt. Gehe dabei vor wie gerade gelernt. Es ist nicht erlaubt, die Binärzahlen für die Multiplikation ins Dezimalsystem umzuwandeln!

Füge hier einen Screenshot deines Codes ein:

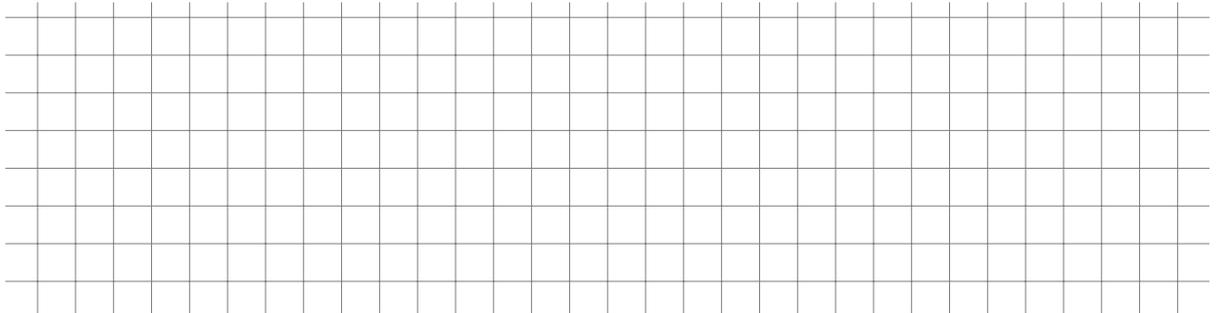
## 4 \*\*Optional: Hexadezimalsystem

In der Computerwelt ist auch das **Hexadezimalsystem**, das Zahlensystem mit Basis 16 und Nennwerten  $0, 1, \dots, 9, A, B, C, D, E, F$  weit verbreitet. Üblicherweise wird der Präfix  $0x$  verwendet, um Hexadezimalzahlen zu kennzeichnen.

### Aufgabe 4.1

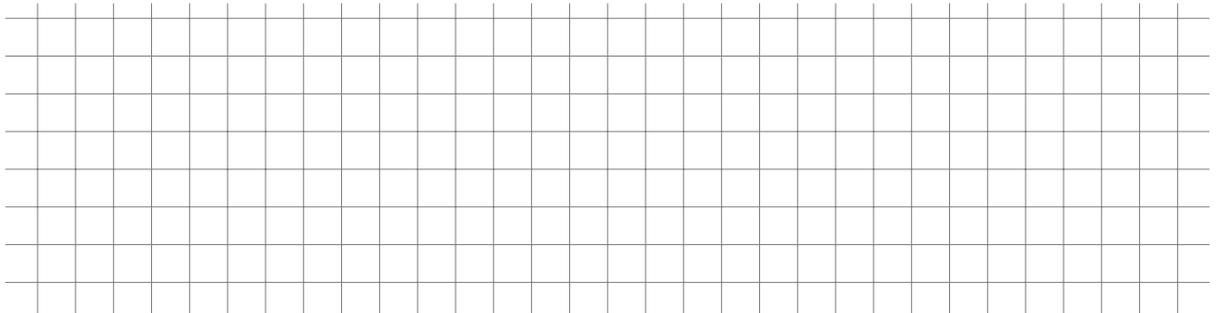
a) Wandle um vom Dezimalsystem ins Hexadezimalsystem:

- $19_{10}$
- $32\,768_{10}$
- $56\,379_{10}$



b) Wandle um vom Hexadezimalsystem ins Dezimalsystem:

- $0x\ C$
- $0x\ FFFF$
- $0x\ B38A$



### Aufgabe 4.2

Zusatzaufgabe: Google und finde heraus, wofür die folgenden Zahlen in 'Hexspeak' stehen.

- $0x\ C0CAC01A$ ,  $0x\ ADD511FE$
- $0x\ DEADC0DE$
- $0x\ DECAFBAD$
- $0x\ FEE1DEAD$
- $0x\ BADCAB1E$

**Aufgabe 4.3**

- a) **Code: Hexadezimalzahl zu Dezimalzahl.** Schreibe eine Funktion `hexa_to_decimal(h)`, die Hexadezimalzahlen ins Dezimalsystem umrechnet und zurückgibt.
- b) **Code: Dezimalzahl zu Hexadezimalzahl.** Schreibe eine Funktion `decimal_to_hexa(d)` für den umgekehrten Schritt: Dezimalsystem  $\rightarrow$  Hexadezimalsystem.

Füge hier einen Screenshot deines Codes ein:

## Lösungen

### Lösung Aufgabe 2.1

a)

$$41\,086_{10} = 4 \times 10^4 + 1 \times 10^3 + 0 \times 10^2 + 8 \times 10^1 + 6 \times 10^0$$

b) Basis: 2, Nennwerte: 0, 1

### Lösung Aufgabe 3.1

a)  $17_{10}$

b)  $11100_2$

c) [https://de.wikipedia.org/wiki/Mano\\_cornuta](https://de.wikipedia.org/wiki/Mano_cornuta)

### Lösung Aufgabe 3.2

a)  $111_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$

b)  $1000011_2 = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 2^6 + 2 + 1 = 64 + 2 + 1 = 67$

c)  $1101010_2 = 106$

d)  $100010001000_2 = 2^{11} + 2^7 + 2^3 = 2184$

### Lösung Aufgabe 3.3

Schicke deinen Code per Teams deiner Lehrperson.

### Lösung Aufgabe 3.4

a) 8 bits mit je 2 möglichen Zuständen:

$$2^8 = 256$$

b)  $2^8 - 1 = 256 - 1 = 255$  (das  $-1$  braucht es, weil 0 die kleinste Zahl ist)

Lösungsvariante 1 im Detail (elegant): Mit 8 Bits kann man  $2^8 = 256$  Zahlen darstellen. Ist 0 die kleinste, muss 255 die grösste sein.

Lösungsvariante 1 im Detail (weniger elegant): Die grösste Zahl ist  $1111\,1111_2$ . Umrechnen in Dezimalzahl ergibt 255

c)  $2^{8x} - 1$

d) 6 Bytes. Mit 5 Bytes kann man maximal eine gute Milliarde darstellen (reicht nicht), mit 6 Bytes sind es 281 Milliarden (reicht).

### Lösung Aufgabe 3.5

a)  $13_{10} = 1101_2$

b)  $19_{10} = 10011_2$

c)  $217_{10} = 11011001_2$

d)  $56\,379_{10} = 1101110000111011_2$

Beispiellösung für eine der Teilaufgaben:

<b>13</b>	
6	1
3	0
1	1
0	1

$$\rightarrow 13_{10} = 1101_2$$

### Lösung Aufgabe 3.6

Schicke deinen Code per Teams deiner Lehrperson.

### Lösung Aufgabe 3.7

a)

$$\begin{array}{r}
 5\ 0\ 9\ 7\ 4 \\
 +\ 8\ 1\ 9\ 6\ 3 \\
 \hline
 1\ 1 \\
 \hline
 1\ 3\ 2\ 9\ 3\ 7
 \end{array}$$

b)

- $1100\ 1100_2 + 0011\ 0011_2 = 1111\ 1111_2$
- $1100\ 1011_2 + 1010\ 1110_2 = 1\ 0111\ 1001$
- $1101\ 0010_2 + 1100\ 0101_2 = 1\ 1001\ 0111_2$
- $1110\ 1111_2 + 1011\ 0101_2 = 1\ 1010\ 0100_2$

Zweites Beispiel im Detail:

$$\begin{array}{r}
 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\
 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \\
 1\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1
 \end{array}$$

Letztes Beispiel im Detail:

$$\begin{array}{r}
 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1 \\
 +\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \\
 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
 \hline
 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0
 \end{array}$$

**Lösung Aufgabe 3.8**

Schicke deinen Code per Teams deiner Lehrperson.

**Lösung Aufgabe 3.9**

- a) Zahl:  $0010\ 1101_2$ , Gegenzahl:  $1101\ 0011_2$
- b)  $0101\ 0101_2 = 85_{10}$ , also  $1010\ 1011_2 = -85_{10}$

**Lösung Aufgabe 3.10**

- a)  $1000\ 0000_2 - 0001\ 1111_2 = 0110\ 0001_2$
- b)  $1010\ 1011_2 - 0110\ 1011_2 = 0100\ 0000_2$
- c)  $1111\ 0010_2 - 1000\ 1111_2 = 0110\ 0011_2$

Erste Aufgabe im Detail: Zwei positive 8-Bit Zahlen  $\rightarrow$  erweitere auf 9-Bit:

$$\begin{aligned}
 1010\ 1011_2 - 0110\ 1011_2 &= 0\ 1010\ 1011_2 - 0\ 0110\ 1011_2 \\
 &= 0\ 1010\ 1011_2 + 1\ 1001\ 0101_2 \\
 &= 10\ 0100\ 0000_2 \\
 &= 0\ 0100\ 0000_2 \\
 &= 0100\ 0000_2
 \end{aligned}$$

Im zweitletzten Schritt ignorieren wir das 10. Bit, da wir mit 9-Bit zahlen gerechnet haben. Im letzten Schritt schreiben wir das Resultat als 8-Bit Zahl, da die ursprünglichen Zahlen auch als solche dargestellt wurden.

**Lösung Aufgabe 3.11**

grösste Zahl:  $0111_2 = 7_{10}$ , kleinste Zahl:  $1000_1 = -8_{10}$

**Lösung Aufgabe 3.12**

Schicke deinen Code per Teams deiner Lehrperson.

**Lösung Aufgabe 3.13**

- a)

$$\begin{array}{r}
 \begin{array}{cccccccc}
 7 & 9 & 0 & 5 & \times & 3 & 8 & 1 & 5 \\
 \hline
 & & & & & 1 & 9 & 0 & 7 & 5 \\
 & & & & & & & 0 & \cdot & \\
 & & & 3 & 4 & 3 & 3 & 5 & \cdot & \cdot \\
 & & 2 & 6 & 7 & 0 & 5 & \cdot & \cdot & \cdot \\
 \hline
 & & 1 & 1 & & 1 & & & & \\
 \hline
 3 & 0 & 1 & 5 & 7 & 5 & 7 & 5 & & 
 \end{array}
 \end{array}$$

