DER COMPUTER UND DIE ZAHLEN

MUSTERLÖSUNGEN

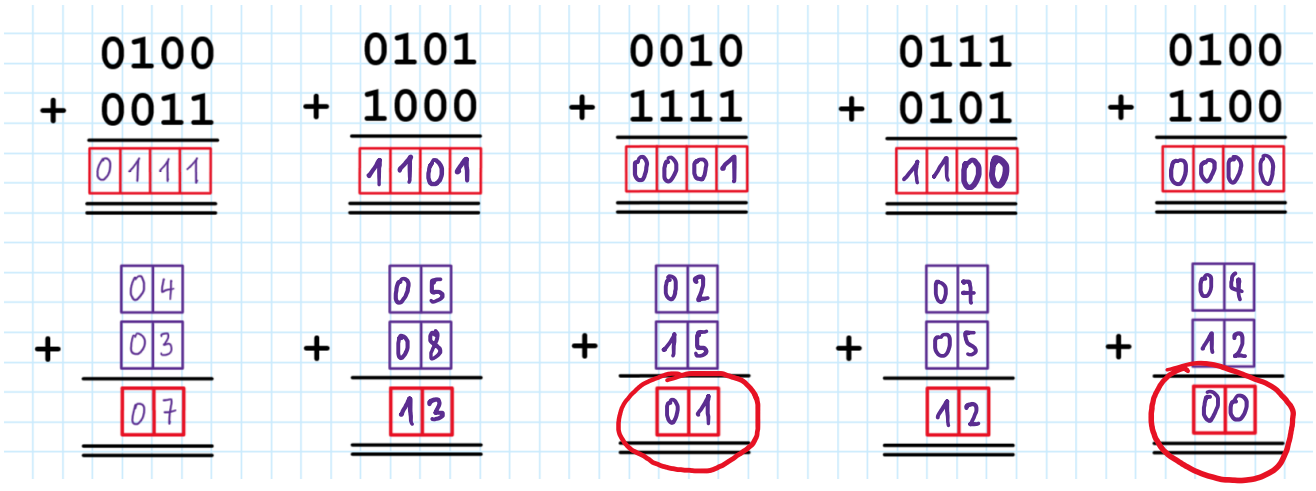
GANZE POSITIVE ZAHLEN

Aufgabe 1: *Ergänze „zum Aufwärmen" in dieser Tabelle die Werte in der Spalte „Anzahl der ganzen Zahlen in Potenzschreibweise" (also in der Form a^n)!*

Architektur	Anzahl der ganzen Zahlen		Datentyp
	in Potenzschreibweise	als Dezimalzahl	
16 Bits	2^{16}	65'536	`short`
32 Bits	2^{32}	4'294'967'296	`int`
64 Bits	2^{64}	1'844'674'473'709'551'616	`long`

Aufgabe 2: *a. Simuliere nun als „Fingerübung" die Arbeit unseres 4-Bit-Prozessors und führe die hier dargestellten Additionen durch!*
Aber Achtung: Nicht alle Additionen werden ein sinnvolles Ergebnis liefern!

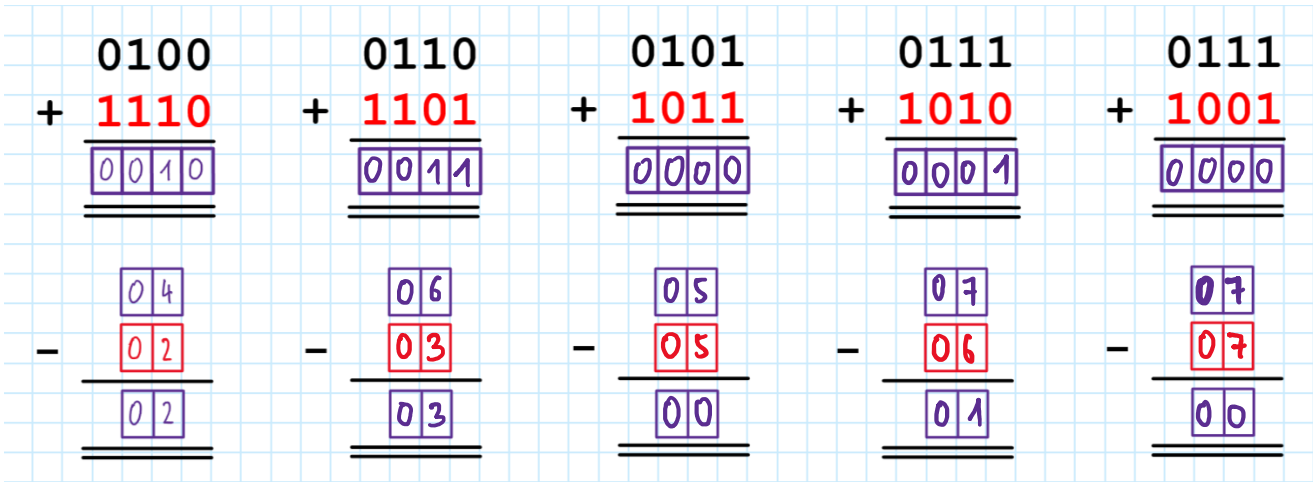
b. Übertrage danach die vierstelligen Binärzahlen in zweistellige Dezimalzahlen und trage diese in die direkt darunter stehenden Additionen ein! Markiere anschliessend diejenigen Rechnungen, die kein sinnvolles Ergebnis liefern!



Aufgabe 3: *a. Führe die hier dargestellten Additionen einer positiven (schwarz gedruckt) und einer negativen Binärzahl (rot gedruckt) wie in Aufgabe 2a durch!*

b. Übertrage danach den ersten Summanden (schwarze Binärzahl) und das Ergebnis (blaue Kästchen) in Dezimalzahlen und trage diese Zahlen in die direkt darunter stehenden Additionen ein (blaue Kästchen)! Berechne anschliessend, welche Dezimalzahl die negativen (roten) Binärzahlen jeweils darstellen (rote Kästchen)!

c. Trage die negativen (roten) Binärzahlen in die unten angefügte Tabelle ein! Kannst du ein Muster erkennen? Versuche Regeln zu beschreiben, nach denen in diesem 4-Bit-Prozessor negative Zahlen gebildet werden!



Dezimalzahl	entsprechende Binärzahl		entsprechende Binärzahl
−1	1111	−5	1011
−2	1110	−6	1010
−3	1101	−7	1001
−4	1100	−8	1000

Die Regeln scheinen so zu sein, dass alle negativen Zahlen **immer mit einer 1** beginnen und dass bei den negativen Zahlen gleichsam **in die entgegengesetzte Richtung gezählt** wird, auch wenn die niedrigste Dezimalzahl -8 mit `1000` den niedrigsten Wert hat und die höchste Dezimalzahl -1 mit `1111` den höchsten Wert hat.

Aufgabe 4: *Besprich mit einer Mitschülerin oder einem Mitschüler deiner Wahl folgende Fragen!*

Wann kommt es bei einer Addition von **zwei positiven Binärzahlen** (die mit einer 0 beginnen) dazu, dass eine **negative Binärzahl** (die mit einer 1 beginnt) **als Ergebnis** herauskommt?
Die **Addition von zwei positiven Binärzahlen** führt genau dann zu einer negativen Binärzahl als Ergebnis, wenn das tatsächliche Ergebnis **grösser ist als die grösstmögliche in einer Prozessor-Architektur darstellbare positive Zahl.** (Bei einer 4-Bit-Architektur wären das alle Zahlen grösser als 7, bei einer 8-Bit-Architektur alle Zahlen grösser als 127, bei einer 16-Bit-Architektur alle Zahlen grösser als 32767, …)
Wie würden die kleinsten/grössten **negativen** Binärzahlen in einer Tabelle bei einem Prozessor mit 8-Bit-Architektur aussehen? Wie sehen sie bei einer 16-Bit-Architektur aus?
Prozessor mit 8-Bit-Architektur: Die kleinste Dezimalzahl -128 wäre die negative Binärzahl `10000000`, die grösste Dezimalzahl -1 wäre die negative Binärzahl `11111111`. Prozessor mit 16-Bit-Architektur: Die kleinste Dezimalzahl -32768 wäre die negative Binärzahl `1000000000000000`, die grösste Dezimalzahl -1 wäre die negative Binärzahl `1111111111111111`.
Wie sieht die **kleinste** darstellbare negative Binärzahl in den verschiedenen Prozessorarchitekturen aus? Wie sieht die Dezimalzahl **-1** jeweils als Binärzahl aus?
Die **kleinste** darstellbare Binärzahl sieht in den verschiedenen Prozessorarchitekturen mit **n Bits** immer so aus, dass sie **mit einer 1 beginnt, gefolgt von n-1 Nullen**. Die Dezimalzahl **-1** sieht in den verschiedenen Prozessorarchitekturen mit **n Bits** immer so aus, dass sie aus **n Einsen** besteht.
Wie könnte ein Algorithmus aussehen, um aus einer positiven Binärzahl eine negative Binärzahl zu machen, also praktisch einen **Vorzeichenwechsel** von + zu - vorzunehmen?
Siehe Blatt 3, Seite 5 der Arbeitsblätter: „Das Zweierkomplement: Der Trick zur Darstellung von negativen Zahlen“

GLEITKOMMAZAHLEN

Aufgabe 5: *a. Wandle die binären Kommazahlen 101.111_2, 110.011_2 und 1001.1001_2 in dezimale Kommazahlen um!*

b. Wandle die dezimalen Kommazahlen 13.78125_{10}, 1.90625_{10} und 0.1_{10} in binäre Kommazahlen um! (Sechs Stellen hinter dem Komma sind genug …)

$$101.111_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$$
$$= 4 + 0 + 1 + 0.5 + 0.25 + 0.125$$
$$= 5.875_{10}$$

$$110.011_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$$
$$= 4 + 2 + 0 + 0 + 0.25 + 0.125$$
$$= 6.375_{10}$$

$$
\begin{aligned}
1001.1001_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \\
&= 8 + 0 + 0 + 1 + 0.5 + 0 + 0 + 0.0625 \\
&= 9.5625_{10}
\end{aligned}
$$

$$
\begin{aligned}
13.78125_{10} &= 1 \cdot 2^3 + 5.78125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 1.78125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1.78125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0.78125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0.28125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0.03125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0.03125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} \\
&\quad + 0.03125 \\
&= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} \\
&\quad + 1 \cdot 2^{-5} + 0 \\
&= 1101.11001_2
\end{aligned}
$$

$$
\begin{aligned}
1.90625_{10} &= 1 \cdot 2^0 + 0.90625 \\
&= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0.40625 \\
&= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0.15625 \\
&= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0.03125 \\
&= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0.03125 \\
&= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \\
&= 1.11101_2
\end{aligned}
$$

$$
\begin{aligned}
0.1_{10} &= 0 \cdot 2^0 + 0.1 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0.1 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0.1 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0.1 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0.0375 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0.00635 \\
&= 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0.00635 \\
&= 0.000110\ldots_2
\end{aligned}
$$

Für die Dezimalzahl 0.1 gibt es als binäre Kommazahl keine endliche Darstellung. Jede Zahl, die man binär mit endlich vielen Kommastellen darstellen kann, kann man auch als Bruch darstellen, dessen Nenner eine Zweierpotenz ist. Die Zahl 0.1 = $\frac{1}{10}$ kann man nicht als Bruch $\frac{a}{2^i}$ darstellen und somit hat sie keine endliche binäre Kommazahldarstellung.

Aufgabe 6: *Bestimme und beschreibe die Informationen, die man benötigt, um die nichtganzen Zahlen 1.5, 1.75 und -1234.567 binär abspeichern zu können!*

Siehe Blatt 4, Seite 7 der Arbeitsblätter: „Die annähernde Darstellung von nichtganzen sowie von sehr grossen Zahlen“

Aufgabe 7: *Beantworte die folgenden Fragen in der Tabelle auf der Rückseite dieses Blattes!*

a. *Berechne (eventuell im Kopf?), welches die jeweils höchsten und niedrigsten Exponenten sind, die bei den Datentypen* `float` *und* `double` *dargestellt werden können! Notiere sie als Dezimalzahlen und stelle sie auch als Bitfolge dar!*

b. *Stelle die grösstmögliche Zahl, die annähernd mit den Datentypen* `float` *und* `double` *dargestellt werden kann, in der binären Darstellung von Gleitkommazahlen dar und berechne (mit Taschenrechner oder Computer) diese Zahl als Dezimalzahl (dargestellt als dezimale Gleitkommazahl)!*
Tipp: $1.11111111111\ldots_2$ kannst du aufrunden auf 10.0_2.

c. *Berechne (mit Taschenrechner oder Computer) die höchste maximale Abweichung (dargestellt als dezimale Gleitkommazahl) von der exakten Darstellung einer Zahl, wenn die Mantisse nach 23 Bits (*`float`*) oder nach 52 Bits (*`double`*) abgeschnitten wird!*

<table>
<tr><th>float</th><th>double</th></tr>
<tr><td colspan="2">höchster und niedrigster Exponent</td></tr>
<tr><td>höchster Exponent:
$01111111_2 = 127$
niedrigster Exponent:
$10000000_2 = -128$</td><td>höchster Exponent:
$01111111111_2 = 1023$
niedrigster Exponent:
$10000000000_2 = -1024$</td></tr>
<tr><td colspan="2">grösste darstellbare Zahl (dargestellt als Binärfolge und als Dezimalzahl in der Form $n \cdot 10^m$)</td></tr>
<tr><td>0 01111111 11111111111111111111111
= 127 ≈ 2

$2 \cdot 2^{127} = 3.40282 \cdot 10^{38}$</td><td>0 01111111111 11111111111111111111111111111111111111111111
= 1023 ≈ 2

$2 \cdot 2^{1023} = 1.7976931348623159 \cdot 10^{313} =$

179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216

(kann weder mit dem Taschenrechner noch mit Excel ausgerechnet werden – aber Python kann es)</td></tr>
<tr><td colspan="2">höchste maximale Abweichung von der exakten Darstellung einer Zahl (dargestellt als $n \cdot 10^m$)</td></tr>
<tr><td>$2^{-23} = 1.19209 \cdot 10^{-7}$</td><td>$2^{-52} = 2.22045 \cdot 10^{-16}$</td></tr>
</table>