Speicherung von Daten am Computer

Wir haben bisher schon besprochen, wie einzelne Zeichen und natürliche Zahlen im Computer gespeichert werden. Auch haben wir die Umwandlung vom Dezimalzahlsystem ins Dualsystem und die Umwandlung vom Dual- ins Dezimalsystem behandelt.

Addition von Dualzahlen

Die Addition läuft genauso ab wie beim schriftlichen Addieren von Dezimalzahlen - siehe Beispiel rechts im Kasten. Das einzige Problem sind die Überträge.

Beispiel:

```
A          = 10011010 (154)
B          = 00110110 (54)
Übertrag   =  11111
              --------------
Ergebnis   = 11010000 (208)
```

Subtrahieren von Dualzahlen

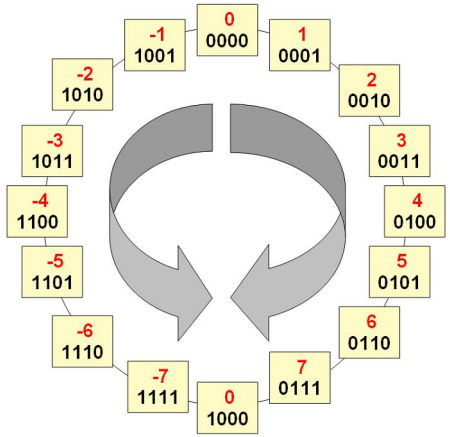
Wenn man zwei Dualzahlen voneinander abziehen möchte, gibt es dafür zwei Umsetzungsmöglichkeiten: Man kann entweder eine komplett neue Rechenlogik aufbauen, die aber wegen des Ausborgens von Zahlen sehr kompliziert ist, oder man führt die Subtraktion auf eine Addition zurück, was technisch deutlich einfacher umzusetzen ist.
Aus der Mathematik wissen wir ja a – b = a + (-b).

Darstellung von negativen ganzen Zahlen

Vor allen nachfolgenden Überlegungen muss man sich zunächst entscheiden, ob man nur natürliche Zahlen speichern möchte oder auch negative Zahlen. Denn nur so kann man die binäre Zahlendarstellung voneinander unterscheiden.

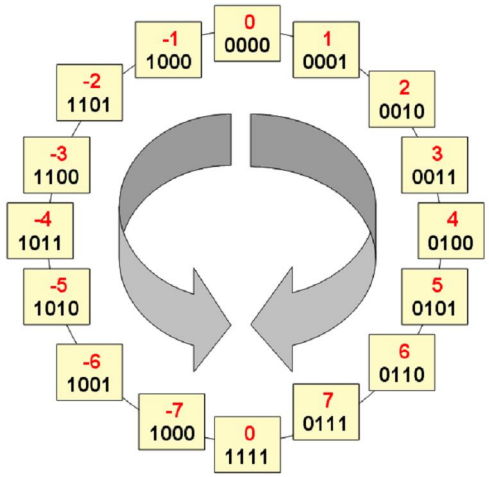
a) Vorzeichenbit

Die naive Vorgehensweise ist es, ein Bit zu verwenden, um das Vorzeichen zu speichern. Ansonsten behält man die restliche Zahlendarstellung bei. Zum Beispiel ist dann **0**001_2= + 1_{10} und **1**001_2= - 1_{10}. Wenn das Bit ganz links 1 ist, ist die Zahl negativ. Wenn das Bit 0 ist, dann ist sie positiv.
Rechts ist diese Zahlendarstellung am Beispiel von 4-Bit-Zahlen abgebildet. Man kann in 4 Bits also Zahlen zwischen -7_{10} und 7_{10} darstellen.
Ein Vorteil der Darstellung ist der einfache Vorzeichenwechsel. Problematisch sind die komplizierte Umsetzung der Addition (z.B. bei -3 + 4) und die nicht eindeutige Darstellung der Null als 0000_2 und 1000_2.

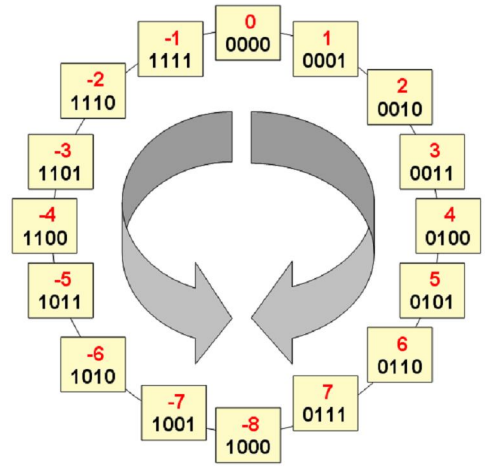


b) Einerkomplement

Eine gewisse Verbesserung erreicht man durch das so genannte Einerkomplement. Man bildet es, in dem man alle Bits invertiert (also umkehrt, aus einer „0“ wird eine „1“ und aus einer „1“ eine „0“).
In vier Bits kann man weiterhin die Zahlen -7_{10} und 7_{10} speichern (siehe nächste Seite). Bei der Verwendung des Einerkomplements muss man bei der Addition keine Unterscheidung mehr zwischen positiven und negativen Zahlen machen und man kann schnell dividieren. Jedoch hat man weiterhin zwei Darstellungen der Null und die Addition ist problematisch, wenn die Null durchlaufen wird.



Einerkomplement bei 4-Bit Zahlen



Zweierkomplement bei 4-Bit Zahlen

c) Zweierkomplement

Das Zweierkomplement basiert auf dem Einerkomplement, löst aber dessen Probleme. Das Zweierkomplement erzeugt man, in dem alle Bits invertiert und 1 addiert.

Beispiel:
Die Gegenzahl von $3_{10} = 0011_2$, also das Zweierkomplement von 3, erhält man, in dem man alle Bits invertiert, d.h. aus 0011_2 wird 1100_2. Anschließend addiert man Eins: $1100_2 + 0001_2 = 1101_2 = -3_{10}$.

Mit 4 Bits kann man durch die Zweierkomplementdarstellung die Zahlen -8 bis +7 darstellen (siehe oben rechts). Bei dieser Darstellung überwiegen die Vorteile: Es ist ein einfacher Vorzeichenwechsel möglich, negative Zahlen werden mit führender 1 dargestellt, die Addition und Subtraktion sind „normal“ und werden nicht unterschieden und man kann einfache Hardware dafür bauen.

Rechnen im Zweierkomplement

Beispiel: -4 + 3 soll berechnet werden. Analog zur Addition bei natürlichen Zahlen erhält man:

```
  1100₂   (= -4)
+ 0011₂   (=  3)
------------------
= 1111₂   (= -1)
```

Aufgabe 1:

Berechne mit Hilfe von 4-Bit-Zahlen:
a) 2 + 3 b) – 6 + 1 c) – 3 – 2 d) – 2 + 2

Aufgabe 2:

Was passiert, wenn man 4 + 4 berechnet?

Alle Zahlen bei den Aufgaben sind im Dezimalsystem angegeben!

In C# werden ganze Zahlen durch folgende Datentypen dargestellt:

Typ	Größe	Darstellbarer Zahlbereich
short	16 Bit	-32 768 ... 32767
int	32 Bit	-2 147 483 648 ... 2 147 483 647
long	64 Bit	-9 223 372 036 854 775 808 … 9 223 372 036 854 775 807

Natürliche Zahlen kann man in C# durch unsigned-Datentypen darstellen, z. B. ushort (Bereich von 0 bis 65535), uint, ulong.

Die Multiplikation und Division von Zahlen kann auf die Addition und Subtraktion zurückgeführt werden. Dieser Text soll nur eine Einführung in die Zahlendarstellung am Computer sein, darum wird an dieser Stelle auf eine ausführliche Darstellung dieser beiden Rechenoperationen verzichtet.

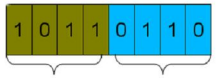
Darstellung von Dezimalzahlen

Die Darstellung von Dezimalzahl am Computer ist nicht ganz einfach, deshalb wird an dieser Stelle auf eine ausführliche Darstellung ebenfalls verzichtet.

Man unterscheidet bei Dezimalzahlen im Computer die Festkommadarstellung und Fließ- bzw. Gleitkommadarstellung:

a) Festkommadarstellung

Eine Dezimalzahl, wie z.B. 11,375 muss mit Nullen und Einsern gespeichert werden. Man teilt dafür ein Byte in einen Vorkomma- und einen Nachkommaanteil auf:



Bei der Festkommadarstellung bleibt diese Einteilung immer fest und ist nicht veränderlich. Um aus dieser Darstellung (also „10110110“) die rationale Zahl zu berechnen, geht man folgendermaßen vor:

Als Vorkommazahl erhält man: $1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0 = 11_{10}$

Als Nachkommazahl erhält man: $0\cdot2^{-1}+1\cdot2^{-2}+1\cdot2^{-3}+0\cdot2^{-4} = 0{,}375$

Als Lösung erhält man schließlich: 11,375

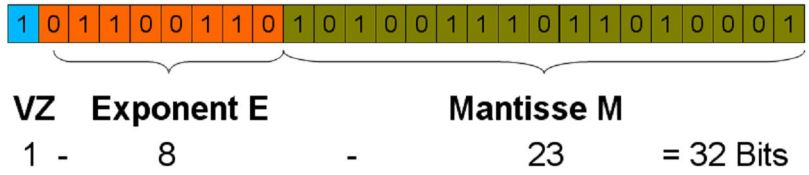
Vorteile der Festkommadarstellung sind die einfache Darstellung und der geringe Rechenaufwand. Nachteilig ist, dass man keine ganz großen bzw. kleinen Zahlen darstellen, da nur eine begrenzte Anzahl von Bits für die Darstellung der Vor- und Nachkommazahl zur Verfügung steht. Die Rechenoperationen **+ - · :** sind überhaupt nicht abgeschlossen, d.h. das Ergebnis kann außerhalb des darstellbaren Bereiches liegen.

b) Fließkommazahlen/Gleitkommazahlen

Die Probleme der Festkommadarstellung werden durch die Gleitkommadarstellung zum größten Teil gelöst, denn jetzt ist die Position des Kommas nicht festgelegt, sondern kann nach Bedarf verschoben werden. Dadurch kann mit gleicher Bitzahl ein größerer Zahlenbereich abgedeckt werden. Leider verliert man dabei auch an Genauigkeit.

Eine Gleitkommazahl lässt sich auf mehrere Arten darstellen. Zum Beispiel kann man die 3 als $3{,}0\cdot10^0$ oder $0{,}3\cdot10^1$ darstellen. Damit die Darstellung eindeutig ist, normalisiert man die Gleitkommazahl, in dem die Mantisse in einen definierten Bereich bringt.

Eine normalisierte 32-Bit-Gleichkommazahl ist folgendermaßen aufgebaut:



Den Wert der Gleitkommazahl erhält man durch folgende Berechnung:

$$(-1)^{VZ}\cdot(1+M_{10})\cdot 2^{E_{10}-Bias}$$

In der Formel steht VZ für das Vorzeichenbit. Wenn es 0 ist, ist die Zahl positiv, wenn es 1 ist, dann ist die Zahl negativ. M_{10} steht für die Mantisse als Dezimalzahl, E_{10} für den Exponent als Dezimalzahl. Um negative Exponenten darstellen zu können, wird von dem Exponenten E_{10} der Bias (bei 32-Bit Zahlen: 127) subtrahiert.

Für das bessere Verständnis führen wir jetzt die Umrechnung an einem Zahlenbeispiel durch. Wir betrachten folgende Gleitkommazahl:

Das Vorzeichenbit VZ = 0. Damit haben wir eine positive Zahl.

Der Exponent ist $E = 1\cdot2^7+1\cdot2^4+1\cdot2^1 = 146_{10}$

Die Mantisse ist

$$M = 1\cdot2^{-1}+1\cdot2^{-4}+1\cdot2^{-5}+1\cdot2^{-6}+1\cdot2^{-9}+1\cdot2^{-13}+1\cdot2^{-14}+1\cdot2^{-17}+1\cdot2^{-20}+1\cdot2^{-21}+1\cdot2^{-22}$$
$$= 0{,}61152052879333496093 7510$$

Den Wert der Zahl erhält aus der obigen Umrechnungsformel:

$(-1)^0\cdot(1+0{,}61152052879333496093 75)\cdot2^{146-127} = 1{,}61152052879333496093 75\cdot2^{19}$
$= 844900{,}875$

In C# kann Dezimalzahlen durch die Datentypen **`decimal`**, **`float`** oder **`double`** darstellen. Der Datentyp **`decimal`** ist wegen seiner höheren Genauigkeit v.a. für Finanzberechnungen geeignet. Wir verwenden in der Regel für Dezimalzahlen den Datentyp double, der mit seiner Genauigkeit für uns völlig ausreichend ist.

Aufgabe 3:

Schreibe ein C# Programm, mit dem man
a) eine Dezimalzahl in eine Dualzahl umwandeln kann
b) eine Dualzahl in eine Dezimalzahl umwandeln kann
Wenn eigene Methoden schon bekannt sind: Die Umwandlungen sollen jeweils in einer eigenen Methode stattfinden. Der Umgang mit der Benutzeroberfläche erfolgt in den Ereignisbehandlungen.

