Die Subtraktion

Der Prozessor ist übrigens so *dumm*, dass er nicht mal „*normal*“ subtrahieren kann, er addiert stattdessen immer eine negative Zahl. Dazu kehrt er das Vorzeichen der zu subtrahierenden Zahl einfach um und addiert diese dann.
Mit unseren 8-Bit lassen sich bekanntlich Zahlen von 0 bis 255 abbilden. Da die CPU nur Binärzahlen verarbeiten kann, stellt sich nun die Frage, wie man, wenn man nur Nullen und Einsen zur Verfügung hat, ein Vorzeichen abbildet?

Negatives Vorzeichen

Die Lösung ist recht einfach, der Prozessor verwendet das höchste Bit (also Bit-7) zur Unterscheidung, ob es sich um eine positive oder negative Zahl handelt. Da wir dieses Bit aber auch benötigen um Zahlen über +127 darzustellen, ist das Vorzeichen von unserer Interpretation abhängig. Im Programm müssen wir also entscheiden, ob wir die Zahl als vorzeichenbehaftet oder als vorzeichenlos betrachten wollen.
Sorry, aber hier ist ein erneuter Ausflug in die Hardware angebracht:
Um das Erkennen des Vorzeichens zu erleichtern, gibt es im Prozessor ein spezielles Flag (das Negativ-Flag). Dieses Flag wird immer dann gesetzt (= 1), wenn wir mit einer Zahl arbeiten, bei der das höchste Bit (Bit-7) gesetzt ist (die Zahl also negativ ist) und gelöscht (= 0) wenn das Bit nicht gesetzt ist. Vereinfacht gesagt, es enthält den gleichen Wert, wie das höchste Bit der Zahl, mit der wir gerade arbeiten.
Da das höchste Bit für die Vorzeichenerkennung benutzt wird können, wir mit unseren 8-Bit also einen vorzeichenbehafteten Wertebereich von -128 bis +127 abbilden.

Werft doch mal einen Blick auf die folgende Tabelle, um zu sehen, wie diese unterschiedliche Interpretation aussieht.

Binär	Hex	ohne Vorz.	mit Vorzeichen
%00000000	$00	0	0
%00000001	$01	1	1
%00000010	$02	2	2
%00000011	$03	3	3
...			
%01111100	$7C	124	124
%01111101	$7D	125	125
%01111110	$7E	126	126
%01111111	$7F	127	127
%10000000	$80	128	-128
%10000001	$81	129	-127
%10000010	$82	130	-126
%10000011	$83	131	-125
...			
%11111100	$FC	252	-4
%11111101	$FD	253	-3
%11111110	$FE	254	-2
%11111111	$FF	255	-1

Die Vorzeichenumkehr (Zweierkomplement)

Wie kann man nun die Vorzeichenumkehr durchführen?
Als Beispiel dient mal wieder die gute alte 42 / `$2A` / `%00101010`. Versuchen wir nun, das Vorzeichen umzukehren, um auf -42 zu kommen.

Da wir schon wissen, dass das höchste Bit zur Kennzeichnung einer negativen Zahl verwendet wird, könnte man nun meinen, dass es reicht dieses einfach zu setzen.

```
 %00101010 <- 42
 %10101010 <- ist das -42???
 --------- >> addieren wir zur Gegenprobe beide Zahlen
 %11010100 => -44, eigentlich sollte (42 + -42) 0 ergeben!
```

Wie wir an der Gegenprüfung, durch Addition der beiden Zahlen sehen, stimmt das nicht, statt 0 kommt -44 raus. Es muss also einen anderen Weg geben.

Die CPU bildet als erstes das sog. Einerkomplement, d. h. wir kehren die Nullen und Einsen einfach um. Dabei wird wieder, wie eben, automatisch das höchste Bit gesetzt, wenn wir von plus nach minus wechseln, bzw. im umgekehrten Fall gelöscht.

```
 %00101010 <- 42
 %11010101 <- Einerkomplement
 --------- >> Addition
 %11111111 => -1, schon dicht dran, aber immer noch nicht 0
```

Wir scheinen der Lösung näher zu kommen, aber unsere Gegenprüfung ist immer noch nicht Null. Das Problem wird durch das Zweierkomplement behoben, dabei wird zum Einerkomplement noch eins addiert.

```
  %00101010 <- 42

  %11010101 <- Einerkomplement
 +%00000001 <- +%1 für Zweierkomplement
 ----------
  %11010110 <- Zweierkomplement
 +%00101010 <- Gegenprüfung durch Addition von +42
 ----------
(1)00000000 => Wir erhalten 0, (den Übertrag können wir ignorieren)
```

Endlich! Wie ~~erhofft~~ erwartet, kommt bei unserer Gegenprüfung 0 heraus. Den Übertrag können wir hier bei unserer 8-Bit-Addition ignorieren. Darum müssten wir uns kümmern, wenn wir z. B. 16-Bit-Operationen implementieren möchten.

Da ich behauptet habe, die CPU kann nur addieren, muss also auch bei einer negativen Zahl durch das Zweierkomplement das Vorzeichen korrekt umgewandelt werden. Wenn man z. B. 42 – -42 rechnen möchte, kehrt der Prozessor (wie oben erwähnt) das Vorzeichen der zweiten Zahl um und addiert diese dann zur ersten. Es wird also 42+42 gerechnet. Kontrollieren wir mal ob die Umkehr des Vorzeichens auch bei einer negativen Zahl funktioniert.

```
  %11010110 <- wir beginnen mit -42

  %00101001 <- Einerkomplement
```

```
+%00000001 <- +%1 für Zweierkomplement
----------
 %00101010 => und wir erhalten wieder 42
```