LU01i - Binär codierte Ganzahlen

Siehe http://www.ulthryvasse.de/index.html

Auch wenn es oft geschrieben wird: Es gibt keine binären Zahlen; Genauso wenig wie es dezimale Zahlen gibt. Korrekt ausgedrückt handelt es sich um die binäre Darstellung oder binäre Codierung von Zahlen.

Und damit wünsche ich viel Erfolg und Spass mit den binären Zahlen in diesem Kapitel. 🤣

Binär codierte Ganzzahlen

Diese Codierung wird unter anderem für Variablen vom Typ `int`, `short` und `long` verwendet. Die Zahlen werden einfach im binären Zahlensystem (siehe auch LU01c - Zahlensysteme) gespeichert und verarbeitet. Auf das Speichern und Verarbeiten von Brüchen wird bewusst verzichtet. Die binär codierten Ganzzahlen können nicht bloss positive Zahlen darstellen, sondern haben den Wertebereich der "Ganzen Zahlen".

Positive Zahlen

Im Speicher sind alle Zahlen als binäre Codes abgelegt. Um diesen binären Code als Dezimalzahl zu lesen, übertragen Sie den Wert einfach ins 10er System.

$1011\ 0010_2 = 178$

Wie das geht, erfahren Sie im Kapitel LU01f - Zahlensysteme umrechnen.

Negative Zahlen als Zweierkomplement

Siehe auch http://de.wikipedia.org/wiki/Zweierkomplement

In der Informatik wird für binäre Ganzzahlen das Zweierkomplement verwendet. Mit dieser Technik können Rechenoperationen ohne spezielle Regeln für positive und negative Zahlen eingesetzt werden. Mehr dazu erfahren Sie im Kapitel LU02e - Mathematische Operationen.

Für positive Zahlen bedeutet das Zweierkomplement nur, dass die erste binäre Stelle Bit '0' sein muss.

Für negative Zahlen gehen Sie wie folgt vor:

1. Addieren Sie 1 zur Zahl: $-13_{10} + 1_{10} = -12_{10}$
2. Entfernen Sie das Vorzeichen der Zahl: $-12_{10} \Rightarrow 12_{10}$
3. Übertragen Sie die Zahl ins Binärsystem: $12_{10} = 0000\ 1100_2$
4. Invertieren Sie alle Bits: $0000\ 1100_2 \Rightarrow$ **$1111\ 0011_2$**

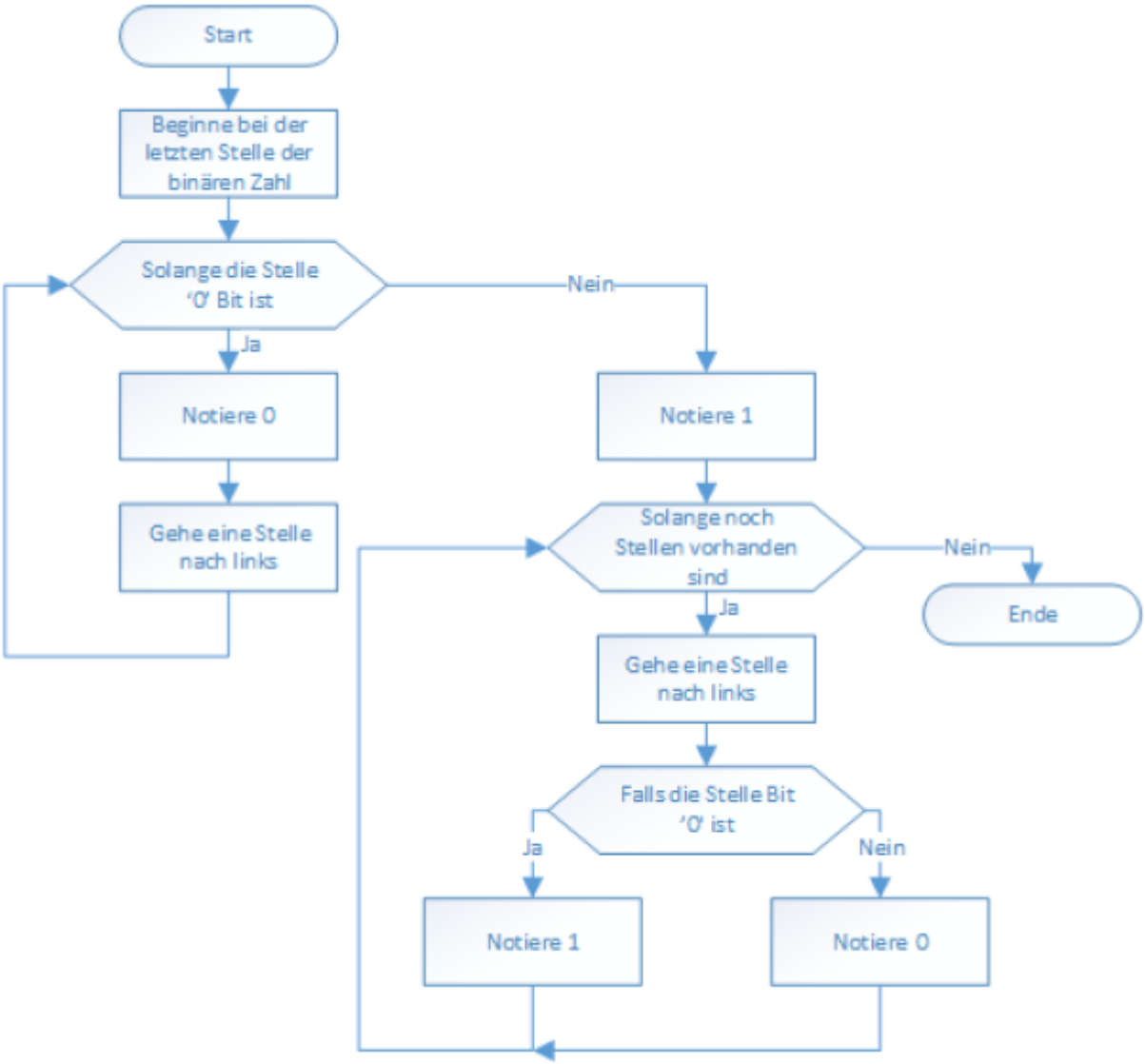
Negative Zahlen haben immer Bit '1' an erster Stelle.

Um eine negative binäre Ganzzahl ins Dezimalsystem zu übertragen, kehren Sie einfach das oben stehende Vorgehen um:

1. Invertieren Sie alle Bits: $1111\ 0011_2 \Rightarrow 0000\ 1100_2$
2. Übertragen Sie die Zahl ins Dezimalsystem: $0000\ 1100_2 = 12_{10}$
3. Fügen Sie das Vorzeichen hinzu: $12_{10} \Rightarrow -12_{10}$
4. Subtrahieren Sie 1 von der Zahl: $-12_{10} - 1_{10} =$ **-13_{10}**

Umwandlung ins Zweierkomplement von Hand

Dieser Programmablaufplan zeigt, wie Sie das Zweierkomplement einer binären Zahl ohne zu rechnen erhalten. Dies stellt eine Alternative zum oben beschriebenen Vorgehen dar:



Wieso kein Vorzeichen?

Auf den ersten Blick scheint die Lösung mit dem Zweierkomplement umständlich. Wäre es nicht einfacher, das erste Bit als Vorzeichen zu verwenden? Zum Beispiel: Das erste Bit ist ...

- '0' ⇒ positive Zahl.
- '1' ⇒ negative Zahl.

Zum Speichern der Zahlen ist diese Lösung problemlos. Weniger als 1 Bit um das Vorzeichen darzustellen, können wir gar nicht verwenden.

Wie sieht es aber beim Rechnen mit diesen Zahlen aus?

6 + 9 --- 15	0000 0110 + 0000 1001 ----------- 0000 1111	12 - 4 ---- 8	0000 1100 - 0000 0100 ----------- 0000 1000
18 + -6 ----- 12	0001 0010 + 1000 0110 ----------- 0000 1100	-5 - 3 ---- -8	1000 0101 - 0000 0011 ----------- 1000 1000

In der ersten Zeile konnten wir noch einfach wie im Dezimalsystem rechnen. Sobald negative Zahlen hinzu kommen, müssen wir aber das Vorzeichen separat behandeln. Wir stellen fest, dass wir unterschiedliche Rechenregeln für positive und negative Zahlen brauchen. Deshalb ist die Lösung mit dem ersten Bit als Vorzeichen nicht sinnvoll.

Wrap around

Mit Wrap around bezeichnen wir das Verhalten, wenn aus einer positiven Zahl plötzlich eine negative Zahl wird. Am Besten lässt sich das Verhalten an einem Beispiel erläutern.

Beispiel

Für unser Java-Beispiel verwenden wir den Datentyp `byte` mit 8 Bit Speicherplatz. In einer Endlosschleife (`while (true)`) addieren wir immer 1 zur Zahl.

```
byte kleineZahl = 124;
while (true) {
    kleineZahl = kleineZahl + 1;
    System.out.println(kleineZahl);
}
```

Ausgabe

```
125
126
127
-128
-127
...
```

Wieso ergibt 127 + 1 = -128? Ein Blick in die binäre Codierung der Zahl lüftet das Geheimnis.

Addiert man zur Zahl 127_{10} (binär 0111 1111_2) Eins dazu, so erhält man $1000\ 0000_2$. Wie Sie gelernt haben, bedeutet die binäre Ziffer 1 an der ersten Stelle, dass es sich um eine negative Zahl handelt. Wir erhalten also -128_{10}.

Im Gegensatz zur Mathematik haben wir also keinen Zahlenstrahl, sondern einen Zahlenkreis.

---

m114-A1G, m114-A1F