LU02e - Mathematische Operationen

Einleitung

Die mathematischen Operationen funktionieren grundsätzlich in allen Stellenwertsystemen auf die gleiche Weise. In diesem Kapitel konzentrieren wir uns auf das Rechnen im Binärsystem. Zur Illustration finden Sie jeweils Beispiele aus dem Dezimalsystem.

Beachten Sie beim Rechnen mit binär codierten Zahlen, dass negative Zahlen als Zweierkomplement gespeichert werden. Wie das geht, haben Sie im Kapitel LU01h - Binär codierte Ganzzahlen erfahren.

Addition

1. Notieren Sie die beiden Zahlen untereinander.
2. Lassen Sie eine Zeile für den Übertrag frei.
3. Addieren Sie die Ziffern der beiden Zahlen und des Übertrags spaltenweise von rechts nach links. Dabei gilt:
   - 0 + 0 + 0 = **0** / Übertrag **0**
   - 0 + 0 + 1 = **1** / Übertrag **0**
   - 0 + 1 + 0 = **1** / Übertrag **0**
   - 0 + 1 + 1 = **0** / Übertrag **1**
   - 1 + 0 + 0 = **1** / Übertrag **0**
   - 1 + 0 + 1 = **0** / Übertrag **1**
   - 1 + 1 + 0 = **0** / Übertrag **1**
   - 1 + 1 + 1 = **1** / Übertrag **1**

```
        Start      Schritt 1  Schritt 2    ...  Schritt 8
Zahl 1  0100 1101  0100 1101   0100 1101   ...  0100 1101
Zahl 2  0010 1001  0010 1001   0010 1001   ...  0010 1001
Übertrag                  1_          01_   ...  0001 001_
Resultat                   0           10   ...  0111 0110
```

```
  0100 1101
+ 0010 1001
-----------
  0001 001    Übertrag
  0111 0110   Resultat
  =========
```

Subtraktion

Anstelle einer Subtraktion führen wir eine Addition aus. Dazu wird der zweite Summand mit -1 multipliziert.

- 753_{10} - 341_{10} ⇒ 753_{10} + (-341_{10})
- 753_{10} - (-341_{10}) ⇒ 753_{10} + 341_{10}

Im Dezimalsystem macht diese Umstellung einer Subtraktion zur Addition wenig Sinn. Der Computer speichert negative binäre Zahlen aber als Zweierkomplement. Durch diese Art der Codierung benötigen Computer keine Logik für Subtraktionen.

Betrachten wir einmal, wie ein Computer die Rechnung 753 - 341 mit 16-Bit binär codierten Ganzzahlen ausführt.

```
Dezimal     Binäre Codierung
   753        0000 0010 1111 0001
+ -341     +  1111 1110 1010 1011
  ----     ----------------------
            1 1111 1101 1100 011    Übertrag
   412        0000 0001 1001 1100   Resultat
   ===        ====================
```

Was geschieht nun mit dem vordersten Übertrag? Diese Ziffer hat keinen Platz mehr und wird deshalb abgeschnitten.

Diese Logik funktioniert auch dann, wenn das Resultat negativ wird:

```
Dezimal     Binäre Codierung
   254        0000 0000 1111 1110
+ -572     +  1111 1101 1100 0100
  ----     ----------------------
            0 0000 0011 1111 100    Übertrag
  -318        1111 1110 1100 0010   Resultat
  ====        ====================
```

Multiplikation

Die Multiplikation mit binär codierten Zahlen ist wesentlich schwieriger. Also haben sich die frühen Pioniere der Informatik gedacht: „Wenn die Multiplikation schwierig ist, dann lassen wir's einfach sein.“ Sie haben richtig gelesen: Ein Computer muss nicht multiplizieren können. Stattdessen verschiebt er die binären Ziffern einfach nach links; Das sogenannte „Links-Shift-Verfahren“.

Notieren wir zunächst die beiden Faktoren der Multiplikation „57 * 13“ im Binärsystem:

```
0000 0000 0011 1001 * 0000 0000 0000 1101 = ???
```

Nun untersuchen wir den zweiten Faktor von rechts (0. Stelle) nach links (15. Stelle).

- Steht eine 1 an der `n.` Stelle:
  - Notieren Sie den 1. Faktor, wobei Sie die Ziffern um n Stellen nach links verschieben.
- Steht eine 0 an der `n.` Stelle:
  - Notieren Sie nichts.

Zum Schluss addieren Sie alle notierten Zahlen. Klingt kompliziert? Vielleicht hilft Ihnen der animierte Ablauf weiter:

leftshift.gif

Division

Die Division von binär codierten Zahlen funktioniert wie die schriftliche Division im Dezimalsystem. Zur Erinnerung betrachten wir zunächst das Prinzip im Dezimalsystem:

```
 639 : 17
 --------
 6      : 17  = 0
 63     : 17  =  3       (3 * 17 = 51)
-51
---
 12
 129    : 17 =    7      (7 * 17 = 119)
-119
----
  10
  100   : 17 =     .5    (5 * 17 = 85)
  -85
  ---
   150  : 17 =        8  (8 * 17 = 136)
  -136
  ----
    140 : 17 = ...
```

Resultat: 639_{10} / 17_{10} = $37.58..._{10}$

Das gleiche Verfahren können Sie für binär codierte Zahlen anwenden.

- Wir ignorieren die führenden Nullen im Dividend und Divisor.
- Anstelle einer Division verwenden wir den Vergleich „>“.
- Anstelle einer Subtraktion verwenden wir immer eine Addition mit dem Zweierkomplement. Dieser erhalten wir, indem wir
  - vom Divisor 1 subtrahieren
  - alle Bits von 0 auf 1 und 1 auf 0 invertieren.

```
  0000 0010 0111 1111 : 0000 0000 0001 0001
  ----------------------------------------
          10011        > 10001 = 1
```

```
        + 101111                          (101111 ist das Zweierkomplement
von 10001)
        --------
          000010
              101       < 10001 = 0       (Hole nächste Ziffer herunter)
              1011      < 10001 = 0
              10111     > 10001 = 1
            +101111
            -------
             000110
                1101    < 10001 = 0
                11011   > 10001 = 1
              +101111
              -------
       Rest    001010  wird verworfen
```

Resultat: 0000 0010 0111 1111$_2$: 0000 0000 0001 0001$_2$ = 0000 0000 0010 0101$_2$ (entspricht 37$_{10}$)

Da wir mit ganzen Zahlen arbeiten, wird der Rest verworfen. Dadurch erhalten wir als Resultat die **nächstkleinere** Ganzzahl.

Multiplizieren statt Dividieren

Anstelle einer Division können wir auch eine Multiplikation mit dem Kehrwert des Divisors durchführen.

$481_{10} / 8_{10} \Rightarrow 481_{10} * (1/8_{10})$

Viele Computersysteme verwenden diese Methode, damit keine zusätzlichen Logikschaltkreise für Divisionen benötigt werden.

Modulo (Division Rest)

Modulo berechnet den ganzzahligen Rest einer Division. Sie können Modulo nur mit zwei Ganzzahlen anwenden.

$639_{10} / 37_{10} = xy_{10}$ **Rest 10**$_{10}$
Somit ist 639_{10} Modulo $17_{10} = 10_{10}$.

Um Modulo von binär codierten Ganzzahlen zu berechnen, führen Sie die Division wie oben beschrieben durch. Anstatt am Schluss den Rest zu verwerfen, bildet dieser Rest das Ergebnis:

```
    0000 0010 0111 1111 Modulo 0000 0000 0001 0001
    ---------------------------------------
           10011        > 10001 = 1
        + 101111                          (101111 ist das Zweierkomplement
```

```
von 10001)
         --------
           000010
               101       < 10001 = 0        (Hole nächste Ziffer herunter)
               1011      < 10001 = 0
               10111     > 10001 = 1
             +101111
             -------
              000110
                 1101    < 10001 = 0
                 11011   > 10001 = 1
               +101111
               -------
        Rest    001010  ist das Ergebnis
```

Resultat: 0000 0010 0111 1111$_2$ Modulo 0000 0000 0001 0001$_2$ = 0000 0000 0000 1010$_2$ (entspricht 12$_{10}$)

---

m114-A1F, m114-A1E