Teilen binär Teil 1 - Vorzeichenlose Ganzzahlen
============

Irgendwann steht jeder Programmieren vor diesem Problem.
Wie teile ich eine Binärzahl durch eine zweite?

Wer in der Grundschule ein wenig aufgepasst hat, sollte in der Lage sein schriftlich eine Zahl durch eine zweite zu teilen.

Hier ein Beispiel:

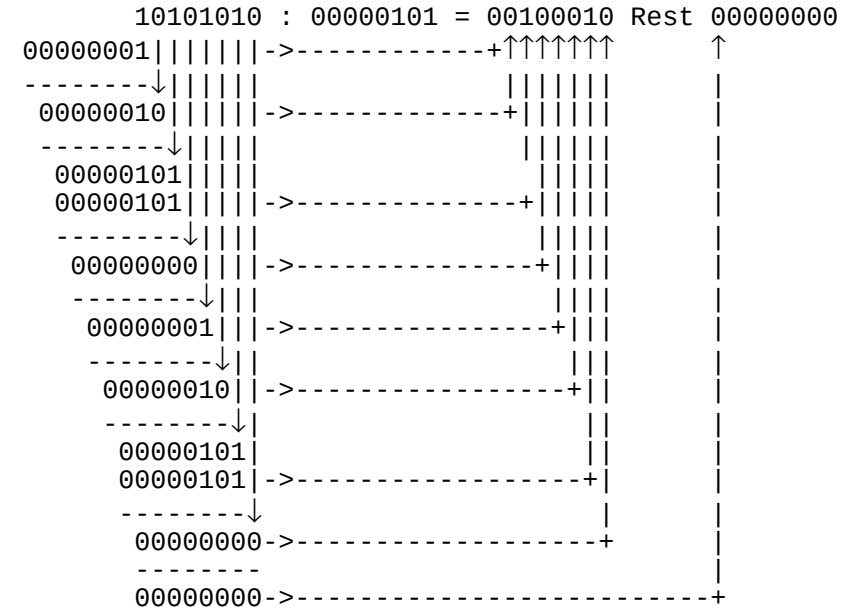
```
 170 : 5 = 34 Rest 0
-15|>--5*3-+↑      ↑
 --↓       |      |
  20       |      |
 -20 >-5*4--+      |
  --               |
   0 ->------------+
```

Nun. Da die Mathematik so universell ist sollte das ganze in Binärdarstellung doch genauso gehen. Probieren wir's mal aus:

```
 170 dez. = 10101010 bin.
   5 dez. =      101 bin.
( 34 dez. =   100010 bin.)
```

```
 10101010 : 101 = 100010 Rest 0
 101|->-----101*1-+↑↑↑↑↑      ↑
 ---↓||||          |||||      |
   00|->----101*0--+||||      |
   --↓|||           ||||      |
    01->----101*0---+|||      |
    --↓||            |||      |
     10->---101*0----+||      |
     --↓|             ||      |
     101|             ||      |
     101->--101*1-----+|      |
     ---↓              |      |
       00->-101*0------+      |
       --                     |
        0->-------------------+
```

Die Sache klappt tatsächlich. Aber DAS in µP-Code um zu setzen scheint doch eine recht vertrackte Sache. Hier 8 Bit Dividend, da drei Bit Divisor, dort 6 Bit Ergebnis und letztlich ein Bit Rest. Wie würde die Sache denn aussehen, wenn ich nur mit 8-Bit-Darstellung das Ganze wiederhole?



Scheinbar klappt das genau so. Aber wie beschreibe ich das so, dass man das auch auf einem µP programmieren kann?

1.) Ich habe dafür genau 8 Einzelschritte benötigt. Also müsste ich wohl eine Schleife bauen.

2.) Ich habe mit dem hochwertigsten Bit 7 begonnen. Dies habe ich in ein Hilfsbyte an die niederwertigste Stelle kopiert.

3.) Dann habe ich geprüft, ob das Hilfsbyte größer oder gleich dem Divisor war. Dem war nicht so und ich habe eine "0" an das hochwertigste Bit des Ergebnisses geschrieben.

4.) Das Hilfsbyte habe ich um ein Bit nach links geschoben und das Bit 6 des Dividenden hinten angehängt. Wieder wie vor geprüft und nochmals eine "0", diesmal an Bit 6, in das Ergebnis gesetzt.

5.) Wieder das Hilfsbyte geschoben, das nächste Bit angehängt und geprüft. Diesmal stand im Hilfsregister 5 und ich habe davon den Divisor 5 abziehen können und diesmal eine "1" im Ergebnis verhaftet.

6.) So habe ich also über alle 8 Bit immer gleich verfahren.

7.) Zuletzt blieb "0" im Hilfsregister. Dies musste der Rest der Division sein. Also hätte ich statt des Hilfsbytes gleich das Byte für den Rest nehmen können.

Bei genauer Betrachtung stellt man fest, dass ich den Dividenden in zwei Schritten ins Hilfsregister Rest schieben könnte. MSBit des Dividenden über LSL ins Carry, Carry über ROL in Rest.

Dann Rest prüfen und ggf. Divisor davon abziehen. Wenn abgezogen werden konnte, eine "1" sonst eine "0" im Ergebnis verhaften.
Auch diese Verhaftung könnte ich mir erleichtern. Im ersten Schritt wurde das Bit 7 als Zielbit gesetzt, im letzten Bit 0. Ich hätte auch immer Bit 0 setzen können und auch jeweils vorher links schieben können. Nach 8 Schritten käme ich auf das Gleiche raus.

Wenn ich also so verfahre, fällt mir eines auf. Ich habe den Dividenden verloren. Er ist am Ende "0". Unschön aber auch wiederum schön. Ich könnte

doch dieses Register für das Ergebnis verwenden. Ich ROLiere es eh vor jedem Schritt. Und sinnvolle Daten würden eh nicht darin zurück gegeben.

Wie sähe jetzt der ganze Ablauf aus?

1.) Lösche Register REST

2.) Erzeuge Schleife über 8 Bit

  3.) Schiebe Bit 7 vom Dividend ins CARRY (LSL). BIT 0 wird dabei "0"

  4.) ROLiere Carry an Bit 0 von Rest

  5.) Rest >= Divisor ?
      JA   - subtrahiere Divisor von Rest,
             Setzte Bit 0 von Dividend auf "1" (INC)
      NEIN - tue nichts.

6.) Bearbeite Schleifenzähler und springe ggf. weiter in Schleife.

War doch eigentlich ganz einfach.

Nun gut, ist nur 8 durch 8 Bit. Aber für viele Anwendungen reicht das schon.

Aber 16 durch 8 ist nicht viel aufwendiger.

1.) Lösche Register REST

2.) Erzeuge Schleife über 16 Bit

  3.) Schiebe Bit 7 vom Divident-LOW ins CARRY (LSL).
      BIT 0 wird dabei "0"
      ROLiere Bit 7 von Divident-HIGH ins CARRY.
      Bit 0 wird hierbei Bit 7 von Divident-LOW.

  4.) ROLiere Carry an Bit 0 von Rest

  5.) Rest >= Divisor ?
        JA   - Subtrahiere Divisor von Rest,
               Setzte Bit 0 von Dividend auf "1" (INC)
        NEIN - tue nichts.

6.) Bearbeite Schleifenzähler und springe ggf. weiter in Schleife.

Das sollte meist genügen. Meist soll eh nur ein Wort vom A/D-Wandler für die LCD-Anzeige umgerechnet werden. Dafür reicht 16div8.

Wie sähe das Ganze jetzt als Code aus. Ich betrachte zuerst mal die Übergabe Dividend und Divisor sowie die Rückgabe Ergebnis und Divisionsrest über Register. Also die "klassische" Methode.

```
; Abschnitt für 8DIV8
;--------------------
.def  Dividend    =R16  ; Zahl, die geteilt werden soll
.def  Ergebnis    =R16  ; erlaubte Mehrfachdefinition
.def  Divisor     =R18  ; Zahl, durch die geteilt werden soll
.def  Rest        =R19  ; Divisionsrest
.def  Schleife    =R20  ; Schleifenzähler

; Abschnitt für 16DIV8
;--------------------
.def  DividendLow =R16  ; Zahl, die geteilt werden soll, Low Byte
.def  ErgebnisLow =R16  ; erlaubte Mehrfachdefinition
.def  DividendHigh=R17  ; Zahl, die geteilt werden soll, High Byte
.def  ErgebnisHigh=R17  ; erlaubte Mehrfachdefinition
;.def Divisor     =R18  ; Zahl, durch die geteilt werden soll
;.def Rest        =R19  ; Divisionsrest
;.def Schleife    =R20  ; Schleifenzähler


Main:
      ;... Division 170 : 5 , 8DIV8
      LDI   Dividend,   170
      LDI   Divisor,    5
      RCALL 8DIV8
      ;...
      ;... Division 8192 : 16 , 16DIV8
      LDI   DividendLow,$00
      LDI   DividendHigh, $20
      LDI   Divisor,    16
      RCALL 16DIV8
      ;...
END

8DIV8:
;=====
      CLR   Rest              ; Divisionsrest löschen
      LDI   Schleife, 8       ; Schleifenzähler 8 BIT setzen
Loop8:
      LSL   Dividend          ; Bit 7 in CARRY
      ROL   Rest              ; dann an Bit 0 von Rest
      CP    Rest, Divisor     ; Test ob Rest >= Divisor
      BRLO  Skip8             ;   und verzweigen
      SUB   Rest, Divisor     ; Abziehen und
      INC   Dividend          ; Bit 0 setzen da von LSL Null
Skip8:
      DEC   Schleife          ; Schleife bearbeiten
      BRNE  Loop8
RET
```

```
16DIV8:
;=====
      CLR   Rest              ; Divisionsrest löschen
      LDI   Schleife, 16      ; Schleifenzähler 16 BIT setzen
Loop16:
      LSL   DividendLow       ; Bit 7 in CARRY
      ROL   DividendHigh      ; durchschieben
      ROL   Rest              ; dann an Bit 0 von Rest
      CP    Rest, Divisor     ; Test ob Rest >= Divisor
      BRLO  Skip16            ;   und verzweigen
      SUB   Rest, Divisor     ; Abziehen und
      INC   DividendLow       ; Bit 0 setzen da von LSL Null
Skip16:
      DEC   Schleife          ; Schleife bearbeiten
      BRNE  Loop16
RET
```

Und nun die Übergabe der Parameter über den STACK. Zugegeben, die Routinen werden größer. Sind aber universeller implantierbar. Und optimieren könnte man das sicher auch noch. SPH gibt's ja meist nicht im verwendeten Prozessor.

```
; Abschnitt für 8DIV8
;--------------------
.def  Temp        =R16  ; Mehr braucht's nicht
.def  Ergebnis    =R16  ; erlaubte Mehrfachdefinition
.def  Rest        =R18  ; Divisionsrest
.def  ZLow        =R30  ; Indexregister Z
.def  ZHigh       =R31

; Abschnitt für 16DIV8
;--------------------
;.def Temp        =R16  ; Mehr braucht's nicht
.def  ErgebnisLow =R16  ; erlaubte Mehrfachdefinition
.def  ErgebnisHigh=R16  ; erlaubte Mehrfachdefinition
;.def Rest        =R18  ; Divisionsrest
;.def ZLow        =R30  ; Indexregister Z
;.def ZHigh       =R31

Main:
      ;... Division 170 : 5 , 8DIV8
      LDI   Temp, 170
      PUSH  Temp
      LDI   Temp, 5
      PUSH  Temp
      RCALL 8DIV8
      POP   Rest
      POP   Ergebnis
      ;... Division 8192 : 16 , 16DIV8
      LDI   Temp, $00
      PUSH  Temp
      LDI   Temp, $20
      PUSH  Temp
      LDI   Temp, 16
      PUSH  Temp
      RCALL 16DIV8
      ;...
      POP   Rest
      POP   ErgebnisHigh
      POP   ErgebnisLow
END
```

```
8DIV8:
;=====
      ; Housekeeping, verwendete Register sichern
      PUSH  Temp              ; könnte beliebiges Register sein
      PUSH  Zhigh             ; Indexregister Z sichern
      PUSH  Zlow
      MOV   Temp, SREG        ; SREG Sichern
      PUSH  Temp              ;  USH SREG geht nicht
      MOV   ZLow, SPL         ; STACK-Pointer kopieren
      MOV   ZHigh, SPH
      ADIW  Z, 6              ; Pointer justieren
      PUSH  R20               ; Wird REST
      PUSH  R21               ; Wird Divisor
      PUSH  R22               ; Wird Dividend
      PUSH  R24               ; Wird Schleifenzähler
      ; Parameter vom STACK holen
      LD    R21, Z+           ; Divisor holen Post INC
      LD    R22, Z            ; dito Dividend
      ; Berechnen
      CLR   R20               ; Divisionsrest löschen
      LDI   R24, 8            ; Schleifenzähler 8 BIT setzen
Loop8:
      LSL   R22               ; Bit 7 in CARRY
      ROL   R20               ; dann an Bit 0 von Rest
      CP    R20, R21          ; Test ob Rest >= Divisor
      BRLO  Skip8             ;   und verzweigen
      SUB   R20, R21          ; Abziehen und
      INC   R22               ; Bit 0 setzen da von LSL Null
Skip8:
      DEC   R24               ; Schleife bearbeiten
      BRNE  Loop8
      ; Ergebnis zurückgeben
      ST     Z, R22           ; Ergebnis
      ST    -Z, R20           ; Divisionsrest Pre DEC
      ; Register Urzustand wieder herstellen
      POP   R24
      POP   R22
      POP   R21
      POP   R20
      POP   Temp
      MOV   SREG, Temp
      POP   Zlow
      POP   Zhigh
      POP   Temp
RET
```

```
16DIV8:
;=====
      ; Housekeeping, verwendete Register sichern
      PUSH  Temp              ; könnte beliebiges Register sein
      PUSH  Zhigh             ; Indexregister Z sichern
      PUSH  Zlow
      MOV   Zlow, SPL         ; STACK-Pointer kopieren
      ADI   Zlow, 6           ; Pointer justieren
      MOV   Temp, SREG        ; SREG sichern
      PUSH  Temp              ;  PUSH SREG geht nicht
      PUSH  R20               ; Wird REST
      PUSH  R21               ; Wird Divisor
      PUSH  R22               ; Wird DividendLow
      PUSH  R23               ; Wird DividendHigh
      PUSH  R24               ; Wird Schleifenzähler
      ; Parameter vom STACK holen
      LD    R21, Z+           ; DivisorHigh holen Post INC
      LD    R22, Z+           ; dito DividendLow
      LD    R23, Z            ; DividendHigh
      ; Berechnen
      CLR   R20               ; Divisionsrest löschen
      LDI   R24, 16           ; Schleifenzähler 16 BIT setzen
Loop16:
      LSL   R22               ; Bit 7 in CARRY
      ROL   R23               ; durchschieben
      ROL   R20               ; dann an Bit 0 von Rest
      CP    R20, R21          ; Test ob Rest >= Divisor
      BRLO  Skip16            ;   und verzweigen
      SUB   R20, R21          ; Abziehen und
      INC   R22               ; Bit 0 setzen da von LSL Null
Skip16:
      DEC   R24               ; Schleife bearbeiten
      BRNE  Loop16
      ; Ergebnis zurückgeben
      ST     Z, R22           ; Ergebnis Low
      ST    -Z, R23           ; Ergebnis High Pre DEC
      ST    -Z, R20           ; dito Divisionsrest
      ; Register Urzustand wieder herstellen
      POP   R24
      POP   R23
      POP   R22
      POP   R21
      POP   R20
      POP   Temp
      MOV   SREG, Temp
      POP   Zlow
      POP   Zhigh
      POP   Temp
RET
```

Der ASCII-Code der Beispiele kann im Acrobat Reader leicht in die Zwischenablage und von dort in den Editor transpheriert werden.

Ob es noch andere Möglichkeiten gibt oder wie das Ganze mit vorzeichen-behafteten Zahlen geht sollte weiteren Beiträgen vorbehalten bleiben.