ÜBUNG: Zeichen zählen ⟶ A-Zähler

Wichtig: Erst Pseudocode-Programm
dann Assemblerprogramm!

⇒ Ein Spaghetti-Programm kann nachträglich meist nicht durch Label / Kommentare strukturiert werden!

AZähler ← 0

Zeichenpositionszeiger (ZPZ) auf 1. Zeichen setzen

repeat

hole Zeichen von ZPZ

ZPZ um ein Zeichen weitersetzen

if Zeichen = 'a' then

AZähler ← AZähler + 1

endif

until Zeichen == 0 (Stringende)

Implementierung: AZähler = r0
ZPZ = r1
aktuelles Zeichen = r2

AsmTest_007

```
    AREA MyData, DATA, align = 2
    GLOBAL MyData,  MeinText

MeinText    DCB     "Das ist ein String mit aaaaa's",0
```

.
.
.
.
.
.
.

```
main    PROC

        mov     r0, #0          ; init. AZähler
        ldr     r1, =MeinText   ; init ZPZ
; 01 -----------------
repeat_01
        ldrb    r2, [r1], #1    ; lade akt. Zeichen, ZPZ++

; 02 -------------------
if_02                           ; ist das Zeichen ein 'a'?
        cmp     r2, #'a'
        bne     endif_02
then_02                         ; Ja, das Zeichen ist ein 'a'.
        add     r0, #1          ; AZähler++
endif_02
; 02 --------------------

until_01
        cmp     r2, #0          ; Stringende erreicht?
        bne     repeat_01
endrepeat_01
; 01 -----------------
```

Pseudocode

Quellzeiger auf 1. Zeichen setzen
Zielzeiger auf 1. Zeichen setzen
while Aktuelles Zeichen aus Quellstring lesen
Aktuelles Zeichen ≠ 0 (= Stringende)
do
'0' in Zielstring schreiben (Init)
① { if Aktuelles Zeichen ≥ '0' then
if Aktuelles Zeichen ≤ '9' then
'1' in Zielstring schreiben
endif
endif
Quellzeiger um 1 weitersetzen
Zielzeiger um 1 weitersetzen
endwhile
0 in Zielstring schreiben (0-Terminator)

Implementierung: Quellzeiger = r2
Zielzeiger = r3
aktuelles Zeichen = r0

---

① if (c ≥ '0' && c ≤ '9') gibt's nicht

AsmTest_008

```
;*****************************************
; Data section, aligned on 4-byte boundery
;*****************************************

    AREA   MyData,  DATA, align = 2
    GLOBAL MyData,  Quellstring, Zielstring

Quellstring   DCB     "aBB12xAuo99",0
Zielstring    SPACE   100    ; reserviere 100 Byte + init. mit 0
```

```
;-----------------------------------------
; main subroutine
;-----------------------------------------
    EXPORT main [CODE]
; -----------------------------------------
; r0: akt. Zeichen
; r2 Quellzeiger (QZ)
; r3 Zielzeiger  (ZZ)
; -----------------------------------------
main    PROC

        ldr     r2, =Quellstring    ; init. Qz
        ldr     r3, =Zielstring     ; init. ZZ
; 01 -----------------
while_01
        ldrb    r0, [r2]            ; Lade akt. Zeichen
        cmp     r0, #0              ; Stringende erreicht ?
        beq     endwhile_01
do_01
        mov     r1, #'0'
; 02 -------------------
if_02                               ; Zeichen >= '0'
        cmp     r0, #'0'
        blo     endif_02
then_02                             ; Ja, Zeichen >= '0'

; 03 -------------------            ; Zeichen <= '9'
if_03
        cmp     r0, #'9'
        bhi     endif_03
then_03                             ; Ja, Zeichen <='9'
        mov     r1, #'1'
endif_03
; 03 -------------------

endif_02
; 02 -------------------

        strb    r1, [r3]            ; Zeichen in Zielstring schreiben
        add     r2, #1              ; Zeiger weitersetzen
        add     r3, #1
        b       while_01
endwhile_01
; 01 -------------------
        mov     r1, #0              ; 0-Term. in Zielstring schreiben
        strb    r1, [r3]
```

Übung: Bitoperationen (1)

Ein ~~auf~~ ab Adresse 0x40000000 liegendes Speicherwort soll wie folgt verändert werden:

Die Bits 0 - 3 sollen auf 1 gesetzt werden,
die Bits 5 - 9 sollen gelöscht (auf 0 gesetzt) werden und
die Bits 12 -15 sollen umgeschaltet (getoggelt) werden.

Schreiben Sie ein Assemblerprogramm.

```
ldr   r0, =0x40000000   ; Speicherwort laden
ldr   r1, [r0]

orr   r1, #0xF          ; Bits 0...3 setzen
(1) bic   r1, #0x3e0    ; Bits 5...9 löschen
(2) eor   r1, #0xF000   ; Bits 12...15 toggeln
str   r1, [r0]          ; Erg. zurückschreiben
```

---

(1) 0x3e0 > FF, aber mit 0x3e << 4 erzeugbar → erlaubt

. .	1 1	1 1 1 0	0 0 0 0
	3	e	0

(2) 0xF000 > FF, aber mit 0xF << 12 erzeugbar → erlaubt

Übung: Bitoperationen (2)

a) Schreiben Sie in Programm "*TestBitPattern*", mit folgendem Verhalten:

[r1] = 1, wenn Bit 0, 3 und 7 von Register r0 auf 1 gesetzt sind und alle anderen Bits auf 0,
= 0 sonst.

Test Bit Pattern:

```
        mov     r1, #0          ;init.
if_01:                  alternativ cmp
        eors    r0, #2_10001001 ;Z=0, wenn nicht gleich
        bne     endif_01        ;überspringen, wenn nicht gleich (Z=0)
then_01:
        mov     r1, #1
endif_01:
```

alternativ:

```
        mov     r1, #0
        eors    r0, #2_10001001
        moveq   r1, #1          @ [r1] = 1, wenn gleiche Bitmuster
```

b) Schreiben Sie in Unterprogramm "*PatternTester*", mit folgendem Verhalten:

[r0] = 1, wenn [r1] und [r2] an denjenigen Bitstellen übereinstimmt die in [r3] mit 1 markiert sind.

= 0 sonst.

r1, r2 Prüfmuster (zu vergleichende Muster)

r3 Maske (interessante Bits = 1)

PatternTester

```
        mov     r0, #0          ; init. Ergebnis
        and     r1, r3          ; uninteressante Bits löschen
        and     r2, r3          ; "
if_02
        eors    r1, r2          ; r1 ≠ r2 ?
        bne     endif_02
then_02
        mov     r0, #1
endif_02
```

alternativ:

```
        mov     r0, #0
        and     r1, r3
        and     r2, r3
        eors    r1, r2
        moveq   r0, #1
```