Aufgabe 8:

6 Punkte

Schreiben Sie ARM Assembler Code um eine Variable mit der Konstanten 3870 zu

multiplizieren, **ohne** hierfür Multiplikationsbefehle einzusetzen. Zu Beginn steht die Variable

im Register R0. Das Ergebnis soll im Register R1 gespeichert werden.

Hinweis: Die Zahl 3870 lässt sich zerlegen in 2 * 15 * 129.

(3 Punkte für die Befehle)

(3 Punkte für die Kommentare)

Aufgabe:

```
rsb     r0,r0,r0,lsl #4          @in r0 steht dann (16-1)*r0=15*r0
```

Aufgabe:

```
rsb     r0,r0,r0,lsl #4         @in r0 steht dann (16-1)*r0=15*r0
add     r0,r0,r0,lsl #7         @in r0 steht dann r0=15*129*r0
```

Aufgabe:

```
rsb     r0,r0,r0,lsl #4     @in r0 steht dann (16-1)*r0=15*r0
add     r0,r0,r0,lsl #7     @in r0 steht dann r0=15*129*r0
mov     r1,r0,lsl #1        @in r1 steht dann r1=15*129*2*r0
```

Aufgabe 9:

12 Punkte

Folgende C Abfrage soll effizient in ARM Assembler umgesetzt werden.

if (a == b || a == 10 || a == 5) **Hinweis:** || steht für logisch oder

c = 1;

else

c = 0;

Für die Variable a ist Register r1, für die Variable b ist Register r2 und für die Variable c ist

Register r0 vorgegeben.

Hinweis: Benutzen Sie bedingte Befehle. Je kompakter (kürzer) Ihr Code ist, desto mehr Punkte können Sie erreichen. Kommentieren Sie Ihren Code.

(6 Punkte für die Befehle)

(6 Punkte für die Kommentare)

label:

mov r0, #0 @ c = 0

mov pc, lr @ Rücksprung

```
label:
mov     r0, #0                  @ c = 0
cmp     r1,r2                   @if a==b
```

```
mov pc, lr @ Rücksprung
```

```
label:
mov     r0, #0                  @ c = 0
cmp     r1,r2                   @if a==b
cmpne   r1,#10                  @wenn nein: if a==10


mov pc, lr @ Rücksprung
```

```
label:
mov     r0, #0                  @ c = 0
cmp     r1,r2                   @if a==b
cmpne r1,#10                    @wenn nein: if a==10
cmpne r1,#5                     @ wenn nein: if a==5

mov pc, lr @ Rücksprung
```

```
label:
mov     r0, #0                  @ c = 0
cmp     r1,r2                   @if a==b
cmpne r1,#10                    @wenn nein: if a==10
cmpne r1,#5                     @ wenn nein: if a==5
moveq r0,#1                     @wenn ==10 oder ==5, c=1
mov pc, lr @ Rücksprung
```

Aufgabe 11:

9 Punkte

Das unten stehende Programm zeigt folgendes Fehlverhalten:

1. Auftretende Kopierfehler werden NICHT korrigiert.

(3 Punkte)

Korrigieren Sie das Programm entsprechend.

2. Das Programm beendet nicht korrekt. Es bleibt hängen oder stürzt ab.

(3 Punkte)

Korrigieren Sie das Programm entsprechend.

3. Machen Sie einige Verbesserungsvorschläge zu dem Programm.

(3 Punkte)

```
.file "tabelle.S"
.text
.align
.global main
.type main,function
main:

adr r0, TAB1]
ldr r1, TAB2
bl kopiere_Tab
adr r0, TAB1
ldr r1, TAB2
bl vergleiche_Tab
mov pc, lr
kopiere_Tab:
ldr r2, [r0]
loop:
ldr r3, [r0],#4
str r3, [r1],#4
subs r2, r2, #1
bne loop
mov pc, lr
vergleiche_Tab:
stmfd sp!, {r4, r5, lr}
ldr r2, [r0]
v_loop:
subs r2, r2, #1
ldmmifd sp!, {r4, r5, pc}
ldr r3, [r0],#4
ldr r4, [r1],#4
cmp r3, r4
beq v_loop
blne korrigiere_Tab
b v_loop
korrigiere_Tab:
stmfd sp!, {lr}
ldr r3, [r0,#0]
str r3, [r1,#0]
ldmfd sp!, {pc}
TAB1:
.word ((TAB1ende-TAB1)/4), 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
TAB1ende:
TAB2:
.word Tabelle2
.comm Tabelle2, TAB1ende-TAB1
```

```
.file "tabelle.S"
.text
.align
.global main
.type main,function
main:

adr r0, TAB1]
ldr r1, TAB2
bl kopiere_Tab
adr r0, TAB1
ldr r1, TAB2
bl vergleiche_Tab
mov pc, lr
kopiere_Tab:
ldr r2, [r0]
loop:
ldr r3, [r0],#4
str r3, [r1],#4
subs r2, r2, #1
bne loop
mov pc, lr
vergleiche_Tab:
stmfd sp!, {r4, r5, lr}
ldr r2, [r0]
v_loop:
subs r2, r2, #1
ldmmifd sp!, {r4, r5, pc}
ldr r3, [r0],#4
ldr r4, [r1],#4
cmp r3, r4
beq v_loop
blne korrigiere_Tab
b v_loop
korrigiere_Tab:
stmfd sp!, {lr}
ldr r3, [r0,#-4]
str r3, [r1,#-4]
ldmfd sp!, {pc}
TAB1:
.word ((TAB1ende-TAB1)/4), 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
TAB1ende:
TAB2:
.word Tabelle2
.comm Tabelle2, TAB1ende-TAB1
```

```
.file "tabelle.S"
.text
.align
.global main
.type main,function
main:
stmfd sp!,{lr}
adr r0, TAB1]
ldr r1, TAB2
bl kopiere_Tab
adr r0, TAB1
ldr r1, TAB2
bl vergleiche_Tab
ldmfd sp!,{pc}
kopiere_Tab:
ldr r2, [r0]
loop:
ldr r3, [r0],#4
str r3, [r1],#4
subs r2, r2, #1
bne loop
mov pc, lr
vergleiche_Tab:
stmfd sp!, {r4, r5, lr}
ldr r2, [r0]
v_loop:
subs r2, r2, #1
ldmmifd sp!, {r4, r5, pc}
ldr r3, [r0],#4
ldr r4, [r1],#4
cmp r3, r4
beq v_loop
blne korrigiere_Tab
b v_loop
korrigiere_Tab:
stmfd sp!, {lr}
ldr r3, [r0,#-4]
str r3, [r1,#-4]
ldmfd sp!, {pc}
TAB1:
.word ((TAB1ende-TAB1)/4), 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
TAB1ende:
TAB2:
.word Tabelle2
.comm Tabelle2, TAB1ende-TAB1
```

```
.file "tabelle.S"
.text
.align
.global main
.type main,function
main:
stmfd sp!,{lr}
adr r0, TAB1]
ldr r1, TAB2
bl kopiere_Tab
adr r0, TAB1
ldr r1, TAB2
bl vergleiche_Tab
ldmfd sp!,{pc}
kopiere_Tab:
ldr r2, [r0]
loop:
ldr r3, [r0],#4
str r3, [r1],#4
subs r2, r2, #1
bne loop
mov pc, lr
vergleiche_Tab:
stmfd sp!, {r4, r5, lr}
ldr r2, [r0]
v_loop:
subs r2, r2, #1
ldmmifd sp!, {r4, r5, pc}
ldr r3, [r0],#4
ldr r4, [r1],#4
cmp r3, r4
beq v_loop
blne korrigiere_Tab
b v_loop
korrigiere_Tab:
stmfd sp!, {lr}      @nicht notwendig !
ldr r3, [r0,#-4]
str r3, [r1,#-4]
mov pc,lr                    @dann aber auch nur mov
TAB1:
.word ((TAB1ende-TAB1)/4), 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
TAB1ende:
TAB2:
.word Tabelle2
.comm Tabelle2, TAB1ende-TAB1
```

12 Punkte

Die Funktion fakultaet(n) berechnet das Produkt aus allen Zahlen von 1 bis n.

Beispiel: fakultaet(4)=1*2*3*4=24

Diese Funktion soll rekursiv berechnet werden:

```
int fakultaet( int n)
{
        if (n==1)
                return(1);
        else
                return(n*fakultaet(n-1));
}
```

Ergänzen Sie die Funktion **fakultaet** in der Form, dass sie rekursiv realisiert wird.

(6 Punkte für die Befehle)

(6 Punkte für die Kommentare)

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}




sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0                       @n in r0




sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0                   @n in r0
        cmp r0,#1                   @n==1




sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0                   @n in r0
        cmp r0,#1                   @n==1
        beq sprung                  @ return (1)



sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0               @n in r0
        cmp r0,#1               @n==1
        beq sprung              @ return (1)
        sub r0,r0,#1            @n-1



sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0               @n in r0
        cmp r0,#1               @n==1
        beq sprung              @ return (1)
        sub r0,r0,#1            @n-1
        bl fakultaet            @Rekursion


sprung:
        ldmfd sp!,{r4, pc}
```

```
main:
        stmfd sp!,{lr}
        mov r0, #5 @ Fakultät von 5 berechnen
        bl fakultaet
        ldmfd sp!,{pc}
fakultaet:
        stmfd sp!, {r4, lr}
        mov r4,r0               @n in r0
        cmp r0,#1               @n==1
        beq sprung              @ return (1)
        sub r0,r0,#1            @n-1
        bl fakultaet            @Rekursion
        mul r0,r4,r0                    @n+n-1
sprung:
        ldmfd sp!,{r4, pc}
```

Aufgabe 6:

2 Punkte

An welcher Adresse wird das Programm fortgesetzt?

Antwort: 0x

```
        mov r0, #20
        cmp r0, r0
        ldreq pc, [pc, r0]
        mov pc, lr
label1: .word 0x0
label2: .word 0x100
label3: .word 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800, 0x900
label5: .word 0x10
```

Aufgabe 6:

2 Punkte

An welcher Adresse wird das Programm fortgesetzt?

Antwort: 0x500

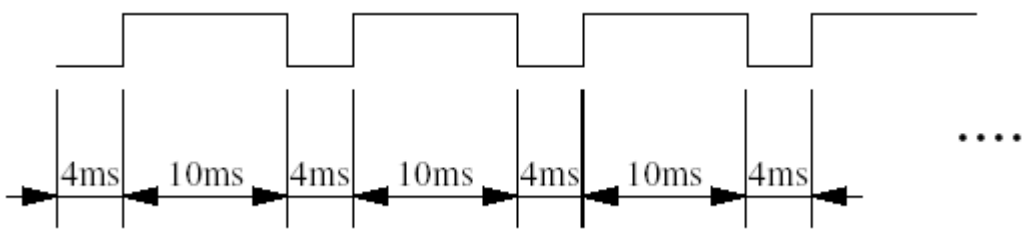
```
        mov r0, #20
        cmp r0, r0
        ldreq pc, [pc, r0]
        mov pc, lr
label1: .word 0x0
label2: .word 0x100
label3: .word 0x200, 0x300, 0x400, 0x500, 0x600, 0x700, 0x800, 0x900
label5: .word 0x10
```

Aufgabe 7:

6 Punkte

Ein Timer soll so initialisiert werden, damit er unten abgebildetes Signal erzeugt. Das System wird mit 5MHz getaktet. Die Initialisierungsroutine ist bis auf die Werte für die Compare-Register fertig.

Berechnen Sie die fehlenden Werte und tragen diese ins Listing ein. Dokumentieren Sie Ihren Lösungsweg.



```
// Timer3 initialisieren
void Timer3_init( void )
{
StructTC* timerbase3 = TCB3_BASE; // Basisadressse TC Block 1
StructPIO* piobaseA = PIOA_BASE; // Basisadresse PIO B
timerbase3->TC_CCR = TC_CLKDIS; // Disable Clock
// Initialize the mode of the timer 3
timerbase3->TC_CMR =
TC_ACPC_CLEAR_OUTPUT | //ACPC : Register C clear TIOA
TC_ACPA_SET_OUTPUT | //ACPA : Register A set TIOA
TC_WAVE | //WAVE : Waveform mode
TC_CPCTRG | //CPCTRG : Register C compare trigger enable
TC_CLKS_MCK2; //TCCLKS : MCKI / 2
// Initialize the counter:
timerbase3->TC_RA = ; //__
timerbase3->TC_RC = ; //__
```

(2 Punkte)

```
// Start the timer :
timerbase3->TC_CCR = TC_CLKEN ;
timerbase3->TC_CCR = TC_SWTRG ;
}
```

Aufgabe 10:

6 Punkte

Zur initialisierung einer seriellen Schnittstelle sollen die ab der Adresse L1 abgelegten hexadezimalen Werte an die ebenfalls dort abgelegten Adressen in einer Schleife abgelegt werden.

Ergänzen Sie die entsprechende Routine.

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
```

```
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
 loop:
    ldr  r2,[r0],#4
```

```
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
 loop:
    ldr  r2,[r0],#4
    ldr  r3,[r0],#4                          @Adresse/Wert in r2/r3
```

```
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
 loop:
    ldr  r2,[r0],#4
    ldr  r3,[r0],#4                    @Adresse/Wert in r2/r3
    str  r3,[r2]                       @ert an Adresse speichern
```

```
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
 loop:
    ldr  r2,[r0],#4
    ldr  r3,[r0],#4                 @Adresse/Wert in r2/r3
    str  r3,[r2]                    @Wert an Adresse speichern
    cmp r0,r1                       @Ende erreicht?
    ____________________________________________________
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

```
@ Funktion
init_ser:
    stmfd sp!, {r0-r3, lr} @ Register retten
    adr r0, L1
    adr r1, L1_end
init_ser_loop:
    ldmia r0!, {r2-r3}
 loop:
    ldr  r2,[r0],#4
    ldr  r3,[r0],#4                 @Adresse/Wert in r2/r3
    str  r3,[r2]                    @Wert an Adresse speichern
    cmp r0,r1                       @Ende erreicht?
    bne  loop
ldmfd sp!, {r0-r3, pc} @ Rücksprung
L1:
.word PMC_BASE+PMC_PCER, 0x4
.word PIOA_BASE+PIO_PDR, 0x18000
.word USART0_BASE+US_CR, 0xa0
.word USART0_BASE+US_MR, 0x8c0
.word USART0_BASE+US_BRGR, 0x29
.word USART0_BASE+US_CR, 0x50
L1_end:
```

Wie muss eine Serielle Schnittstelle programmiert werden, damit bei einem Prozessortakt von 25 MHz eine asynchrone Übertragung von 38400 kBaud realisiert wird?

USCLK[0] = ?

USCLK[1] = ?

CD = ?