Strukturierte Assemblerprogrammierung

Übungsaufgaben zur Vorbereitung:

1. Ein Feld von Daten beginne bei Adresse 0x20000000:

   ```
   Bytefeld    DCB       11, 'B', 0xB, 0b01000010
   ```

   a) Welcher Wert steht für "Bytefeld" in der Symboltabelle?
   b) Wie lautet die Assemblersequenz, um das erste Byte (die 11) in das Register r0 zu laden?
   c) Was steht in r1 und r2 (Hex.) nach folgender Assemlersequenz?

   ```
   ldr   r2, =Bytefeld
   ldrh  r1, [r2]
   ```

2. In Register r0 soll das Datenwort 0xAB00 geladen werden. Geben Sie den Befehl an, wenn der mov-Befehl dafür verwendet werden soll.

3. In Register r0 soll das Datenwort 0x1256ABCD geladen werden. Geben Sie den Befehl an.

4. Welcher Sprungbefehl muss verwendet werden, damit ein Sprung nach Lab1 erfolgt, wenn [r0] >-15 ist?

   ```
   mov     r1,  #-15
   cmp     r0,  r1
   _____   Lab1
   …
   Lab1:
   ```

5. In r0 steht eine **vorzeichenlose** Zahl. Welcher Sprungbefehl muss verwendet werden, damit dann ein Sprung nach Lab2 erfolgt, wenn [r0] <= 33 ist?

   ```
   cmp     r0,  #33
   ____    Lab2
   …
   Lab2:
   ```

6. In welchem Wertebereich (von … bis ...) muss [r0] liegen, damit ein Sprung nach Lab3 erfolgt?

   ```
   ldr     r1,  =0xFFFFE000
   ands    r0,  r1
   beq     Lab3
   …
   Lab3:
   ```

7. Welchen Wert muss [r0] haben, damit ein Sprung nach Lab4 erfolgt?

```
mov   r1, #0xFA
eors  r0, [r1], LSL #8
beq   Lab4
…
Lab4:
```

8. Was steht nach der folgenden Assemblersequenz in r0?

```
ldr   r1, =0xFF0000BB
and   r0, r1, #0xA7
```

9. Was steht nach der folgenden Assemblersequenz in r0?

```
ldr   r1, =0xFF00BB1A
orr   r0, r1, #0xA700
```

10. Was steht nach der folgenden Assemblersequenz in r0?

```
ldr   r1, =0xFF0000BB
eor   r0, r1, #0xA70
```

11. Geben Sie eine Assemblersequenz an, um die Bits 0-3 in r0 zu löschen und die Bits 4-7 zu setzen. Die anderen Bits (8-31) sollen unverändert bleiben.

12. Geben Sie eine Assemblersequenz an, so dass ein Sprung nach Lab5 genau dann erfolgt, wenn die Bits 8-11 in r0 auf 1gesetzt sind (unabhängig davon, wie die anderen Bits gesetzt sind).

```
_____
_____

beq   Lab5
…
Lab5:
```

Strukturierte Assemblerprogrammierung

Aufgabe a: Strukturierte Assemblerprogrammierung

Es ist ein Assemblerprogramm zu entwickeln, das eine Zahlentabelle (*DataList*) aufsteigend sortiert. Die Länge der Tabelle soll veränderbar sein.

```
;************************************************
; Data section, aligned on 16-byte boundery
;************************************************

        AREA MyData, DATA, align = 4

DataList        DCD     35, -1, 13, -4096, 511, 101, -3, -5, 0, 65
DataListEnd     DCD     0

        GLOBAL DataList
        GLOBAL DataListEnd
```

Das Assemblerprogramm soll mit Strukturierungslabels (vgl. Vorlesung Kap. 8) auf strukturierte Weise realisiert werden.

Themen der Aufgabenstellung:

- Implementierung einfacher Kontrollstrukturen durch Abbildung auf bedingte und unbedingte Sprünge.
- Implementierung einfacher Zahlenvergleiche durch das Setzen von Status-Bits mit Vergleichsbefehlen sowie deren Auswertung durch bedingte Sprungbefehle.

Vorzubereiten:

- Algorithmus verstehen
- Abbildung strukturierter Anweisungen auf Assemblercode (vgl. Vorl. Kap. 8)
- verwendete Adressierungsarten (vgl. Vorl. Kap. 6)
- Vergleichsoperatoren und Flags (vgl. Vorl. Kap. 8)
- Assemblerdirektiven (vgl. Vorl. Kap. 7)

Strukturierte Assemblerprogrammierung

Lösungsverfahren: Bubblesort

Das Verfahren beruht darauf, dass der zu sortierende Zahlenbereich mehrmals vom Anfang bis zum Ende durchlaufen wird. Bei jedem Durchlauf werden immer zwei benachbarte Zahlen verglichen. Wenn die beiden betrachteten Zahlen nicht der Sortierreihenfolge genügen, werden sie vertauscht.

Das Durchlaufen wird abgebrochen, wenn bei einem Durchlauf keine Vertauschung mehr vorgenommen wurde.

---------- Pseudocode des Algorithmus ----------

```
Getauscht ← Ja
while  Getauscht == Ja  do
        Getauscht ← Nein
        Zeiger auf den ersten Wert setzen
        while  Zeiger zeigt nicht auf den letzten Wert  do
                if  aktueller Wert > folgender Wert  then
                        Tabelleneinträge tauschen
                        Getauscht ← Ja
                end if
                Zeiger auf den folgenden Eintrag setzen
        end while
end while
```

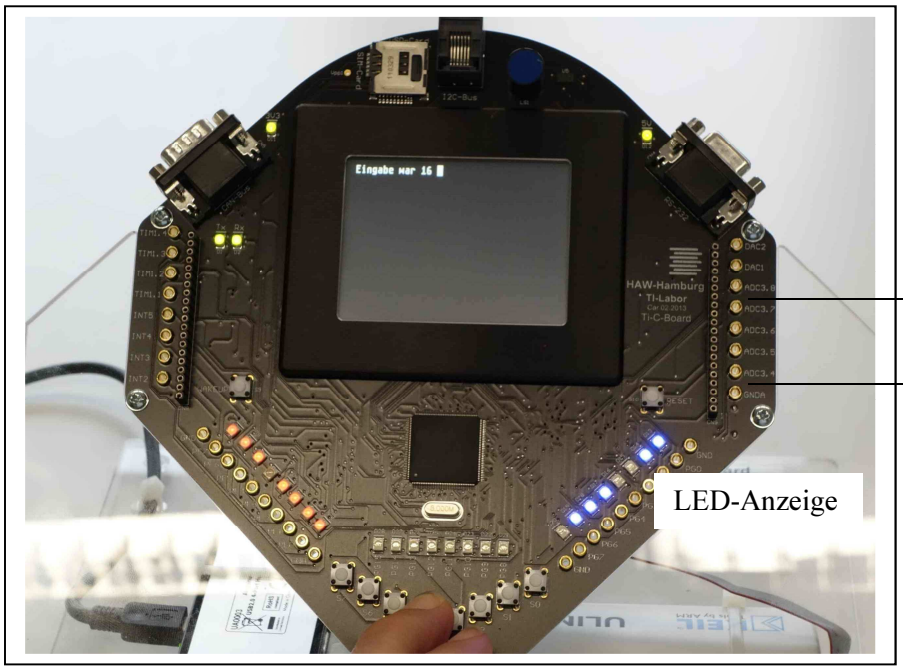
Verwenden Sie Strukturierungslabels der Form

WHILE_01 ….. DO_01 …… ENDWHILE_01
IF_03 …... THEN_03 ….. ELSE_03 ….. ENDIF_03

(vgl. Vorl. Kap. 8). Kommentieren Sie aussagekräftig.

Strukturierte Assemblerprogrammierung

Aufgabe b: Interface-Programmierung



Es ist ein Assemblerprogramm zu entwickeln, welches die Spannung am Analog-Digital-umsetzer A3.7 misst (0V - maximal 3V) und auf den Leuchtdioden (LEDs) an den Ports PG0-PG15 als Leuchtbalken mit variabler Länge ausgibt.

Bei 0V sollen alle LED aus sein, bei 3V sollen 15 LED an sein,
d.h. pro 0.2V geht eine LED an.

Eingangsbuchsen: ADC 3.7 und GNDA
LED-Ausgabe: Port PG0 - PG15

Das System wird mit C-Funktionen initialisiert, wobei diese in der zur Verfügung gestellten main.s (vgl. EMIL) mit dem Schlüsselwort EXTERN eingebunden werden. Sie starten die Batch-Prozedur auf dem Desktop und ersetzen das im erzeugten Projekt enthaltene main.s oder übertragen die Inhalte.

Programmbeispiel:

Das bei EMIL vorgegebene Programm `main.s` gibt den mit dem 12-Bit-AD-Umsetzer eingelesenen Spannungswert auf den Ausgabeports PG0…PG7 als **Binärzahl** aus, d.h. also nicht als Leuchtbalken, wie in der Aufgabe gefordert. Es ist wie folgt realisiert:

```
; Diese EQU-Direktiven zur Adressenvereinbarung nur für die ASM-Implementierung
; mit dem TI-Board nutzen. Ersatz für die korrespondierenden EXTERN Vereinbarungen,
; die die Initialisierung auch für die Simulation ohne Board unterstützen.
ADC3_DR            equ     0x4001224C
PERIPH_BASE        equ     0x40000000
AHB1PERIPH_BASE    equ     (PERIPH_BASE + 0x00020000)
GPIOE_BASE         equ     (AHB1PERIPH_BASE + 0x1000)
GPIOG_BASE         equ     (AHB1PERIPH_BASE + 0x1800)
GPIO_G_SET         equ     GPIOG_BASE + 0x18
GPIO_G_CLR         equ     GPIOG_BASE + 0x1A
;
; RN: Rename-Direktive, ordnet CPU-Registern Variablennamen zu.
adc_wert           RN   7        ; Wert!!!
adc_dr             RN   8        ; Adresse!!
gpio_set           RN   9
gpio_clr           RN   10


;*****************************************
; Code section, aligned on 256-byte boundery
;*****************************************

     AREA    MyCode, CODE, readonly, align = 8
     GLOBAL  main


main   PROC

;  I/O-Adressen in Registern speichern
     ldr     adc_dr,    =ADC3_DR          ; Adresse des ADC
     ldr     gpio_clr,  =GPIO_G_CLR       ; I/O löschen
     ldr     gpio_set,  =GPIO_G_SET       ; I/O setzen

messschleife
     ldr     adc_wert, [adc_dr]           ; Messwert lesen


; Ausgabewert ermitteln
     mov     R3, adc_wert                 ; ADC-Wert lesen (12 Bit)
     mov     R4, R3, LSR #4               ; und auf 8 Bit reduzieren



; LED Ausgabe
     mov     R5, #0xffff
     strh    R5, [gpio_clr]               ; LEDs löschen
     strh    R4, [gpio_set]               ; Ausgabe Bitmuster

     mov     R0, #0x20
     bl      Delay
     b       messschleife


forever b    forever                      ; nowhere to return if main ends
```

Ändern und ergänzen Sie das Programm so, dass es die Aufgabenstellung erfüllt.

Strukturierte Assemblerprogrammierung

Vorgehensweise:

- Verwenden Sie keine „magic numbers“,
- wählen Sie aussagekräftige Namen für Konstanten,
- kommentieren Sie das Programm aussagekräftig.

Lösungshinweise und Anforderungen:

1. Es soll über 16 Spannungsmessungen gemittelt werden, bevor der Wert angezeigt wird. Verwenden Sie hierfür z.B. eine Zählschleife nach folgendem Schema (vgl. Vorl. Kap. 8):

   ```
   for_01 … until_01 … do_01 … step_01 … enddo_0
   ```

   Die DELAY-Funktion soll dabei mit der Mittelung wie eine Tiefpassfilterung wirken, die Messstörungen abschwächt.

2. Die Länge des anzuzeigenden LED-Balkens (Balkenlänge) mit 16 LEDs steht als Binärzahl in den vorderen 4 Bit des vom AD-Umsetzers gelesenen 12-Bit-Wertes (Bit 11 - 8).

3. Den Binärwert zur Ausgabe auf den LEDs erhält man mit: **(1 << Balkenlänge) – 1**
   Beispiel: 0000000000000001 um Balkenlänge=5 nach links geschoben ergibt
   0000000000100000 davon 1 subtrahiert ergibt
   0000000000011111 ➔ Balken der Länge 5

4. Die Messungen und die Anzeige sollen in einer Dauerschleife laufen.