Strukturierte Assemblerprogrammierung

Name : _____	**Name : _____**
Vorname : _____	**Vorname : _____**
Matrikel-Nr : _____	**Matrikel-Nr : _____**

Aufgabenstellung:

- die folgenden Vorbereitungsaufgaben bearbeiten
- die Aufgabe a) vor dem Praktikum fertig stellen, Aufgabe b) vor dem Praktikumstermin möglichst weit vorbereiten
- die weiteren Aufgaben im Praktikum fertigstellen

Themen zur Aufgabenstellung:

- strukturierte Programmierung in Assembler
- Zugriff auf Analog-Digitalwandler-Ausgaben
- Adressierungsarten

Hinweise zur Aufgabenbearbeitung:

Die Bearbeitung von Aufgabe a) ist im zweiten Termin und von Aufgabe b) spätestens zum nächsten (dritten) Termin abzugeben und mit mir zu diskutieren.

Register- /Speicherinhalte sollen in der Regel **im Hexformat** angegeben werden.

Strukturierte Assemblerprogrammierung

Übungsaufgaben zur Vorbereitung:

1. Mit welchem Befehl kann Register r0 mit 2 (4, 8, 16,...) multipliziert werden? Mit welchem Befehl durch 2 (4, 8, 16,...) dividiert werden? LSL r0,r0, #1
   LSL r0,r0, #2 etc.
2. Das Feld Messdaten beginne bei Adresse 0x1000. Geben Sie den Speicherinhalt an den Adressen 0x1000, 0x1001, 0x1002, 0x1003, 0x1004, 0x1005, 0x1006, 0x1007 an!

   ```
   Messdaten   DCW   0x1234, 0x56
   Parameter   DCD   0x00FF, 0xFF00
   ```

   34,12,56,00,FF,00,00,FF00
3. Ein Feld von Daten beginne bei Adresse 0x40000000:

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

   a) Welcher Wert steht für "Bytefeld" in der Symboltabelle? 0x40000000
   b) Wie lautet die Assemblersequenz, um das erste Byte (die 11) in das Register r0 zu laden? ldr r1, =Bytefeld
   ldrb r0, [r1]
   c) Was steht in r1 und r2 (Hex.) nach folgender Assemblersequenz?

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

   ~~00004000~~ 40000000
   00000B42
4. In Register r0 soll das Datenwort 0xAB00 geladen werden. Geben Sie den Befehl an, wenn der mov-Befehl dafür verwendet werden soll. mov r0, #0xAB00
5. In Register r0 soll das Datenwort 0x1256ABCD geladen werden. Geben Sie den Befehl an.
   ldr r0, =#0x1256ABCD
6. Welcher Sprungbefehl muss verwendet werden, damit dann ein Sprung nach Lab1 erfolgt, wenn [r0] >-15 ist?

   ```
   mov     r1,  #-15
   cmp     r0,  r1
   bgt     Lab1
   …
   Lab1:
   ```
7. 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
   bls     Lab2
   …
   Lab2:
   ```

Strukturierte Assemblerprogrammierung

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

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

0x0000 ... 0x0FFF

0x1FFF

9. Welchen Wert muss r0 haben, damit ein Sprung nach Lab4 erfolgt?

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

0xFA00000000

10. Was steht nach der folgenden Assemblersequenz in r0?

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

0x000000A3

11. Was steht nach der folgenden Assemblersequenz in r0?

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

0xFF00BF17

12. Was steht nach der folgenden Assemblersequenz in r0?

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

0xFF000ACB

13. 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.

#2_1000
ldr r1, #0b11
bic r0, r1

#2_11110000
ldr r1, #0b11110000
orr r0,r1

14. 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).

#2_000000000000
_____ ldr r1, #0b111100000000
_____ ands r1,r0,r1

```
beq    Lab5
…
Lab5:
```

Strukturierte Assemblerprogrammierung

Aufgabe a): Strukturierte Assemblerprogrammierung

Es ist ein Assemblerprogramm zu entwickeln, mit dessen Hilfe eine Zahlentabelle (*DataList*) aufsteigend sortiert werden kann. 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, 8224, 101, -3, -5, -310, 0, 65
DataListEnd    DCD  0

        GLOBAL DataList
        GLOBAL DataListEnd
```

Das Assemblerprogramm soll mit Hilfe von Strukturierungslabels (s. Vorlesung) 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 Hilfe von Vergleichsbefehlen sowie deren Auswertung durch bedingte Sprungbefehle.

Vorzubereiten:

- Algorithmus verstehen
- Abbildung strukturierter Anweisungen auf Assemblercode (s. Vorlesung)
- verwendete Adressierungsarten (s. Vorlesung)
- Vergleichsoperatoren und Flags
- Assemblerdirektiven (s. Vorlesung)

Strukturierte Assemblerprogrammierung

Lösungsverfahren: (Bubblesort)

Das Verfahren beruht darauf, dass der Zahlenbereich mehrmals vom Anfang bis zum Ende durchlaufen wird. Bei jedem Durchlauf werden immer zwei benachbarte Zahlen verglichen. Wenn die 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
```

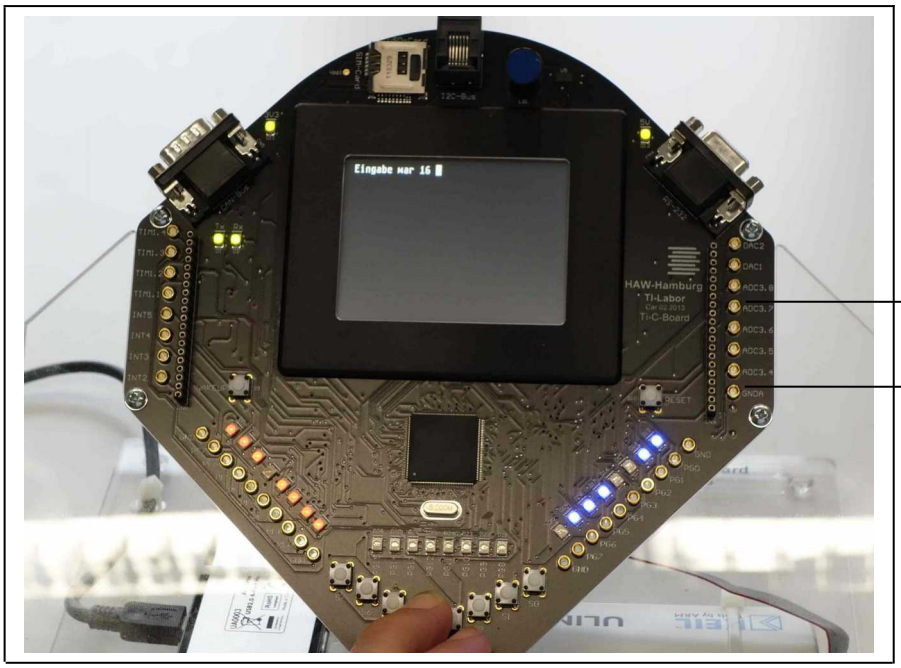
Anm.: Verwenden Sie Strukturierungslabels der Form

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

o.ä. (vergl. Vorlesung). Kommentieren Sie aussagekräftig.

Strukturierte Assemblerprogrammierung

Aufgabe b): Hardwareprogrammierung, Sensordaten verarbeiten



Es ist ein Assemblerprogramm zu entwickeln, welches die Ausgangsspannung eines Sensors am Analog-Digitalwandler A3.7 misst (0V - maximal 3V) und auf den Leuchtdioden (LED) an den Ports PG0-PG15 als mehr oder weniger langen Leuchtbalken 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

Programmbeispiel:

Strukturierte Assemblerprogrammierung

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

```
   EXTERN Init_TI_Board         ; Initialize the serial line
   EXTERN ADC3_CH7_DMA_Config     ; Initialize the ADC
   EXTERN Delay                 ; Delay (ms) function
   EXTERN GPIO_G_SET            ; Set output-LEDs
   EXTERN GPIO_G_CLR            ; Clear output-LEDs
   EXTERN ADC3_DR         ; ADC Value (ADC3_CH7_DMA_Config has to be called before)

;*****************************************
; Data section, aligned on 4-byte boundery
;*****************************************
   AREA MyData, DATA, align = 2
;*****************************************
; Code section, aligned on 8-byte boundery
;*****************************************
   AREA |.text|, CODE, READONLY, ALIGN = 2

; RN: Direktive, um Registern ‘Namen’ zu geben
adc_wert          RN   7       ; Wert!!!
adc_dr            RN   8       ; Adresse!!
gpio_set          RN   9
gpio_clr          RN   10

;-------------------------------------------
; main subroutine
;-------------------------------------------
   EXPORT main [CODE]

main PROC

     BL      Init_TI_Board               ; Initialize the serial line to TTY
     BL      ADC3_CH7_DMA_Config         ; Initialize and config ADC3.7

     ;  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 loeschen
     STRH    r4, [gpio_set]              ; Ausgabe Bitmuster

     MOV     r0, #0x20
     BL      Delay

     B       messschleife

forever B    forever                    ; nowhere to retun if main ends
```

Strukturierte Assemblerprogrammierung

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

Vorgehensweise:

- Verwenden Sie keine „magic numbers“ im Programm,
- 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.
   Anm.: Verwenden Sie hierfür z.B. ein Zählschleife nach folgendem Schema (s. Vorl.):
   `for_01 … until_01 … do_01 … step_01 … enddo_0`

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

3. Den Binärwert zur Ausgabe auf den LED 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