

**ARM-Cortex-M4 / Thumb-2-Befehlssatz**  
**Adressierungsarten und arithmetische Operationen**

**Aufgabenstellung:**

- das beigefügte Assembler-Programm schrittweise ausführen
- sich mit der Handhabung der Entwicklungswerkzeuge vertraut machen
- mit dem Debugger die Belegung von Hauptspeicher und Registern ermitteln

**Themen zur Aufgabenstellung:**

- Speicherorganisation im Hauptspeicher und in den Registern
- Transportbefehle
- Addition von signed und unsigned Integer-Werten
- Adressierungsarten

**Hinweise zur Aufgabenbearbeitung:**

***Die Bearbeitung ist spätestens bis zum nächsten Termin abzugeben und mit mir zu diskutieren.***

Die ermittelten Register- und Speicherinhalte sind in das beigefügte Schema einzutragen.

**!!! WICHTIG:** Geben Sie alle Register- /Speicherinhalte **im Hexformat** an. **!!!**

## Übungsaufgaben zur Vorbereitung

1. Geben Sie das 8-bit-Zweierkomplement der folgenden Zahlen an:

- a) +72
- b) -77

2. Geben Sie die Dezimalwerte der folgenden vorzeichenlosen 8-bit-Binärzahlen an:

- a) 1101 1110
- b) 0011 1111

3. Geben Sie die Dezimalwerte der folgenden 8-bit-Zweierkomplement-Zahlen an:

- a) 1101 1111
- b) 0011 1111

4. Addieren Sie binär und geben Sie die Flags an:

a)

b)

5. Subtrahieren Sie durch „Addition des Zweierkomplements“ und geben Sie die Flags an:

a) 00111110 - 01011111

b) 10000010 - 01000001

6. Geben Sie die Codierung des folgenden (0-terminierten) ASCII-Strings an (im Hex.-Format): "DA da 04",0

## Initialisierungsteil des Assemblerprogramms

```
; Schreibkonvention für Schlüsselwörter im ARM-Assembler (Keil uVision)
; Groß-/Kleinschreibung egal, aber nicht 'mixed' im Schlüsselwort
; Konvention h i e r: Direktiven GROSS, Parameter für Direktiven: klein
;
; label:      Anfang immer in Spalte 0, case sensitive
; Direktiven: mindestens ein blank am Anfang
; Konstanten: binär: 2_10111000... / dezimal: 12345... / hexadezimal: 0xaffe...
;
;*****
; Initialisierte globale Daten (Data-RAM mit Startadresse 0x20000000)
;*****
        AREA MyData, DATA, align = 4      ; align !=!! : p a r a m e t e r für den Block MyData,
                                           ; Grenze des B l o c k s: modulo 4 = 2^2
                                           ; 16-Byte-Alignment wg. Darstellung im Memory-Fenster

        GLOBAL MyData, MeinNumFeld, MeinHaWoFeld, MeinTextFeld, MeinByteFeld, MeinBlock

; DCD: 32 Bit (word) / DCW: 16 Bit (halfword) / DCB: Byte
MeinNumFeld    DCD    0x22, 2_00111110, -52, 78, 0x60000000, 0x50000000

MeinHaWoFeld   DCW    0x1234, 0x5678, 0x9abc, 0xdef0

MeinTextFeld   DCB    "ABab0123",0      ; Nullterminierung bei Strings

        ALIGN    4                      ; hier nur Didaktik wg. besserer Darstellung im Memory-Fenster

MeinByteFeld   DCB    0xef, 0xdc, 0xba, 0x98
;*****
; nicht Initialisierte globale Daten (Data-RAM)
;*****
        ALIGN    4                      ; empfehlenswert für hohe Performance, wenn in MeinBlock
MeinBlock      SPACE 0x20                ; Worte oder Halbworte abgelegt werden
```

Geben Sie die RAM-Belegung (im Hex-Format) an: ab Adresse 0x20000020  
(**&**MeinNumFeld in das Memory-Fenster eingeben ).

Zur Erinnerung: Little Endian Prozessor

Adresse	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x20000020																
0x20000030																
0x20000040																
0x20000050																

## Laden von Konstanten in Register

- 01 Konstanten der Form  $m \cdot 2^N$  mit  $m=0 \dots 255=0xFF$  und  $N=0 \dots 31$  (Links-Schiebefaktor) können mit **mov** direkt in ein Register geladen werden oder mit **add**, **sub**,... benutzt werden.  
( $m=0 \dots 2^{16}-1=0xFFFF$  ohne shift geht beim Cortex auch, aber nur bei mov, nicht bei add, sub,...)

---

```
mov r0,#0x12 ; Anw-01
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 01 an.

- 
- 02 Der ARM-Assembler erlaubt auch die Angabe negativer Konstanten, z.B. **mov R1, #-10**.  
Der Assembler ersetzt dann diesen Befehl durch  
**mvn R1, #10-1** bzw.  
**mvn R1, #9** (Anm.: Einerkomplement = Zweierkomplement - 1).  
 Mit **mvn** wird das *Einerkomplement* einer Konstante der Form  $m \cdot 2^N$  mit  $m=0 \dots 255$  und  $N=0 \dots 31$  (Links-Schiebefaktor) in ein Register geladen. Durch Subtraktion einer 1 vor der Negation wird das *Zweierkomplement* (= Einerkomplement + 1) der Konstante abgelegt.

---

```
mov r1,#-2 ; Anw-02
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 02 an.
2. Vergleichen Sie die programmierte Anw.-2 (s. Editor) mit der tatsächlich vom ARM-Assembler assemblierten Anweisung (s. Disassembler).

(Disassembler).

- 
- 03 Andere Konstanten müssen zuvor im Speicher abgelegt werden (z.B. am Ende des Programms). Von dort können Sie dann mit Hilfe relativer Adressierungsarten zugegriffen werden. Da dies mühevoll zu programmieren ist, bietet der ARM-Assembler einen „Pseudobefehl“ an, z.B. **ldr R2, =0x12345678**.

Der Assembler sorgt dann dafür, dass die Konstante im Speicher abgelegt wird und der Pseudobefehl ersetzt wird z.B. durch **ldr R2, [PC, #0x54]**, d.h. der Prozessor greift relativ zum Program Counter auf die im Speicher abgelegte Konstante zu.

---

```
ldr r2,=0xfe543210 ; Anw-03
```

1. Optional: Auf welcher Adresse A liegt die Konstante (s. Disassembler)?
2. Optional: Geben Sie den Speicherinhalt bei Adresse A an (s. Memory-Viewer).
3. Geben Sie den Registerinhalt nach Ausführung von Anw. 03 an.

Adresse A = \_\_\_\_\_ (Hex.)

---

## Laden von Variablen in Register

04 Der ARM hat keine direkte Adressierung (*Lade Inhalt von Adresse xxxxx*). Aus diesem Grund wird erst die Adresse des zu lesenden Speichers als Konstante geladen. Anschließend kann indirekt auf die Speicheradresse zugegriffen werden.

*Beispiel:* „MeineVariable“ sei die symbolische Adresse des zu lesenden Datenfeldes.  
Die Variable kann dann mit der folgenden Befehlssequenz gelesen werden:

```
ldr    r0, =MeineVariable
ldr    r1, [r0]
```

---

```
ldr    r0, =MeinByteFeld    ; Anw-04
```

1. Optional: Vergleichen Sie die programmierte Anw. 04 (s. Editor) mit dem assemblierten Befehl (s. Disassembler).
2. Geben Sie den Registerinhalt nach Ausführung von Anw. 04 an.

---

05 Auch bei Byte und Halbwortzugriffen (z.B. : **ldrb r1, [r0]** ) wird immer das ganze Register verändert. Nicht verwendete Stellen werden durch 0 aufgefüllt.

---

```
ldrb   r1, [r0]              ; Anw-05
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 05 an.

06...07    Beim *little-endian*-Byte-Ordering ist die Bytereihenfolge in Halbworten und Worten im Speicher vertauscht (Beachte: Nur bei Halbworten und Worten und nur im Speicher!).

---

ldrh    r2, [r0]                                ; Anw-06  
ldr     r3, [r0]                                ; Anw-07

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 06 und Anw. 07 an.
2. Erklären Sie das Ergebnis.

---

## Feldzugriffe mit konstantem Offset

08...10    Auf die Elemente von Datenfeldern (Arrays) kann z.B. mit Hilfe eines Basisadressregisters (Startadresse des Feldes) und einem Offset zugegriffen werden.

---

ldr    r4, =MeinHaWoFeld                    ; Anw-08  
ldr    r5, [r4]                                ; Anw-09  
ldr    r6, [r4, #4]                            ; Anw-10

1. Geben Sie die Registerinhalte nach Ausführung von Anw. 08 ... 10 an.

## Variablen speichern

11 ...17    Beim Speichern von Halbworten und Worten ist das Alignment für hohe Performance empfehlenswert.

```
-----  
ldr  r0, =0x123456ab      ; Anw-11  
ldr  r1, =MeinBlock       ; Anw-12  
str  r0, [r1]              ; Anw-13  
str  r0, [r1, #4]          ; Anw-14  
  
mov  r2, #0x1a             ; Anw-15  
strb r2, [r1, #9]          ; Anw-16  
strb r2, [r1, #10]         ; Anw-17
```

1. Geben Sie die Startadresse A von „MeinBlock“ an.
2. Geben Sie den Speicherinhalt ab „MeinBlock“ nach Ausführung von Anw. 11 ... 17 an.

Adresse „MeinBlock“ = \_\_\_\_\_ (Hex.)

---

## Ganzzahladdition und Flags

18 ...21

```
-----  
-  
ldr      r0, =MeinNumFeld    ; Anw-18  
ldr      r1, [r0]             ; Anw-19  
ldr      r2, [r0, #4]         ; Anw-20  
adds     r3, r1, r2           ; Anw-21
```

1. Geben Sie die Registerinhalte und Flags nach Ausführung von Anw. 18 ... 21 an.
2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?
3. Angenommen die Operanden sind vorzeichenbehaftet, war die Addition fehlerfrei?





22 ...25

```
-----  
ldr      r0,=MeinNumFeld+8      ; Anw-22  
ldr      r1, [r0]                ; Anw-23  
ldr      r2, [r0, #4]           ; Anw-24  
adds     r3, r1, r2              ; Anw-25
```

1. Geben Sie die Registerinhalte und Flags nach Ausführung von Anw. 22 ... 25 an.
2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?
3. Angenommen die Operanden sind vorzeichenbehaftet, war die Addition fehlerfrei?

26 ... 29

```
-----  
ldr  r0,=MeinNumFeld+16      ; Anw-26  
ldr  r1, [r0]                 ; Anw-27  
ldr  r2, [r0, #4]            ; Anw-28  
adds r3, r1, r2               ; Anw-29
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 26 ... 29 an.
2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?
3. Angenommen die Operanden sind vorzeichenbehaftet, war die Addition fehlerfrei?

## Feldzugriffe mit Offset und Update

30 ... 32

```
-----  
ldr      r0,=MeinTextFeld      ; Anw-30  
ldrb     r1, [r0, #1]!          ; Anw-31  
ldrb     r1, [r0, #1]!          ; Anw-32
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 30 ... 31 an.

2. Geben Sie den Registerinhalt nach Ausführung von Anw. 32 an.

33 ... 35

```
-----  
ldr      r0,=MeinHaWoFeld      ; Anw-33  
ldr      r2, [r0], #4           ; Anw-34  
ldr      r2, [r0], #4           ; Anw-35
```

1. Geben Sie den Registerinhalt nach Ausführung von Anw. 33 ... 34 an.

2. Geben Sie den Registerinhalt nach Ausführung von Anw. 35 an.