

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

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

Aufgabenstellung:

- das beigelegte Assembler-Programm schrittweise ausführen
- sich mit der Handhabung der Entwicklungswerzeuge 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.

In grau sind optionale Aufgaben gegeben, die nicht bearbeitet werden müssen.

Die ermittelten Register- und Speicherinhalte sind in das beigelegte 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 0100 1000
b) -77 1011 0011

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

- a) 1101 1110 222
 b) 0011 1111 63

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

- a) 1101 1111 -33
 b) 0011 1111 63

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

mit Vorzeichen richtig

vorzeichenbehaftet falsch >127

a) $\begin{array}{r}
 00101110 \\
 + 01011011 \\
 \hline
 - & - & - & - \\
 1000 & 1001
 \end{array}$

Vorzeichenlose Rechnung richtig? Vorzeichenbehaftete Rechnung richtig?

b)
$$\begin{array}{r}
 11111110 \\
 + 11011111 \\
 \hline
 11011101
 \end{array}$$

Vorzeichenlose Rechnung richtig? Vorzeichenbehaftete Rechnung richtig?

mit Vorzeichen korrekt

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

a) $00111110 - 01011111$

$$\begin{array}{r}
 00111110 \\
 + \quad \cdots \cdots \cdots \\
 - \quad 1010\ 0001 \\
 \hline
 1101\ 1111
 \end{array}$$

Vorzeichenlose Rechnung richtig?
Vorzeichenbehaftete Rechnung richtig?

mit Mannschaft bewältigt.

b) 10000010 - 01000001

10000010	C =
+ -----	V =
- 1011 1111	N =
0100 0001	

Vorzeichenlose Rechnung richtig?
Vorzeichenbehaftete Rechnung richtig?

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

44 41 20 64 61 20 30 34 00

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 0x200000XX (zur Bestimmung von „XX“ & 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
0x200000 <u>28</u>	2 2	00	00	00	3E	00	00	00	CC	FF	FF	FF	4E	00	00	00
0x200000 <u>38</u>	0 0	00	00	60	00	00	00	50	34	12	78	56	BC	9A	F0	DE
0x200000 <u>48</u>	4 1	42	61	62	30	31	32	33	00	00	00	00	EF	DC	BA	98
0x200000 <u>58</u>	0 0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Laden von Konstanten in Register

- 01 Konstanten der Form $m \cdot 2^N$ mit $m=0\dots255=0xFF$ und $N=0\dots31$ (Links-Schiebefaktor) können mit **mov** direkt in ein Register geladen werden oder mit **add**, **sub**,... benutzt werden.
($m=0\dots2^{16}-1=0xFFFF$ 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.

Register R0			
00	00	00	12
31	16	15	0

- 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 $m=0\dots255$ und $N=0\dots31$ (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. Optional: Geben Sie den Registerinhalt nach Ausführung von Anw. 02 an.
2. Optional: Vergleichen Sie die programmierte Anw.-2 (s. Editor) mit der tatsächlich vom ARM-Assembler assemblierten Anweisung (s. Disassembler).

Register R1				=	Dezimalwert bei vorzeichenbehafteter Interpretation: _____
FF	FF	FF	FE	0	Dezimalwert bei vorzeichenloser Interpretation: <u>4.294.967.294</u>

- 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 (s.u.) nach Ausführung von Anw. 03 an.

Adresse A = 0x08000244 (Hex.)

Speicherinhalt bei Adresse A				Register R2 nach Anw-03			
A	A+1	A+2	A+3	31	16	15	0
10	32	54	FE	FE	54	32	10

Laden von Variablen in Register

- 04 Der ARM hat keine direkte Adressierung (*Lade Inhalt von Adresse xxxx*). 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:

```
Idr      r0, =MeineVariable  
Idr      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.

Register R0							
31	16	15	0	54	00	00	20

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

ldr r1, [r0] ; Anw-05

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

Register R1							
31	16	15	0	EF	00	00	00

- 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!).
-

```
Idrh r2, [r0] ; Anw-06
Idr r3, [r0] ; Anw-07
```

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

Register R2				Register R3			
31	16	15	0	31	16	15	0
00	00	DC	EF	98	BA	DC	EF

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.
-

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

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

Register R4			
31	16	15	0
20	00	00	40

Register R5				Register R6			
31	16	15	0	31	16	15	0
56	78	12	34	DE	F0	9A	BC

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“ = 0x20000058 (Hex.)

Speicherinhalt ab „MeinBlock“

AB	56	34	12	AB	56	34	12	00	1A	1A	
A	A+1	A+2	A+3							

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.

Register R1

00	00	00	22
31	16	15	0

Register R2

00	00	00	3E
31	16	15	0

Register R3

00	00	00	60
31	16	15	0

Negativ N = 0.....
Carry C = 0.....
Overflow V = 0.....

2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?

3. Optional: Angenommen sie 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.

Register R1

FF	FF	FF	CC
31	16 15		0

Register R2

00	00	00	4E
31	16 15		0

Register R3

00	00	00	1A
31	16 15		0

Negativ	N = 0.....
Carry	C = 1.....
Overflow	V = 0.....

2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?

3. Optional: Angenommen sie 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.

Register R1

20	00	00	38
31	16 15		0

Register R2

50	00	00	00
31	16 15		0

Dezimal

Register R3

B0	00	00	00
31	16 15		0

Negativ	N = 1.....
Carry	C = 0.....
Overflow	V = 1.....

2. Angenommen die Operanden sind vorzeichenlos, war die Addition fehlerfrei?

3. Optional: Angenommen sie sind vorzeichenbehaftet, war die Addition fehlerfrei?

Feldzugriffe mit Offset und Update

30 ... 32

```

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

```

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

Register R0				Register R1			
31	16	15	0	31	16	15	0
20	00	00	49				42

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

Register R0				Register R1			
31	16	15	0	31	16	15	0
20	00	00	4A				61

33 ... 35

```

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

```

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

Register R0				Register R2			
31	16	15	0	31	16	15	0
20	00	00	44				34

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

Register R0				Register R2			
31	16	15	0	31	16	15	0
20	00	00	48				BC