

LOAD, STORE, bedingte Befehle,  
Speicherbereiche



**h\_da**

HOCHSCHULE DARMSTADT  
UNIVERSITY OF APPLIED SCIENCES

**fbi**  
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

WS2017

**Termin 4**

LOAD, STORE, bedingte Befehle, Speicherbereiche

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

**Ziele:**

Verständnis für LOAD und STORE Befehle, bedingte Befehle und die verschiedenen Speicherbereiche. Ziel ist die Implementierung mit möglichst geringer Codegröße sowie der Umgang mit einem Debugger/Simulator und der Entwicklungsumgebung.

**Arbeitsverzeichnis:**

Kopieren Sie sich das Verzeichnis, welches Ihnen im Praktikum zur Verfügung gestellt wird, in Ihr persönliches Verzeichnis. Dort stehen Ihnen dann alle benötigten Dateien zur Verfügung.

**Vorbereitung**

Arbeiten Sie sich in die Gruppe der LOAD und STORE Befehle, bedingte Befehle und Verzweigungsbefehle am Beispiel der folgenden Befehle des ARM-Prozessors ein:

Instruktion	Bedeutung
ADDNE R1, R2, #1	R1 := R2 + 1, falls das Z-Bit im Prozessorstatuswort <b>nicht</b> gesetzt ist
LDR R1, [R2]	R1 := mem <sub>32</sub> [R2]
LDREQ R1, [R2]	R1 := mem <sub>32</sub> [R2], falls das Z-Bit im Prozessorstatuswort gesetzt ist
LDRB R1, [R2]	R1 := mem <sub>8</sub> [R2]
STR R1, [R2]	mem <sub>32</sub> [R2] := R1
STRB R1, [R2]	mem <sub>8</sub> [R2] := R1
ADR R1, Marke	R1:=PC+(Offset zur Marke)
B Marke	PC wird auf Adresse der Marke gesetzt
BEQ Marke	PC wird auf Adresse der Marke gesetzt, falls das Z-Bit im Prozessorstatuswort gesetzt ist
BNE Marke	PC wird auf Adresse der Marke gesetzt, falls das Z-Bit im Prozessorstatuswort <b>nicht</b> gesetzt ist
LDR R1, = Marke	R1 := mem <sub>32</sub> [PC+(Offset zur Hilfsmarke)] , dies ist eine Pseudoinstruktion

**Aufgabe 1:**

Auf welchen Adressen wird der Inhalt von Register r1 gespeichert? Ergänzen Sie sie Kommentarzeilen.

```

mov    r0, #0
str    r1, [r0], #4    // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
eor    r0, r0, r0
str    r1, [r0, #4]    // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
mov    r0, #0
str    r1, [r0]!       // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
sub    r0, r0, r0
str    r1, [r0, #4]!   // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
and    r0, r0, #0
strb   r1, [r0, #1]!   // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
mov    r1, #4
strb   r1, [r0, r1]!   // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____

```

**Aufgabe 2:**

Bearbeiten Sie schriftlich die Fragen.

- a) Auf welche Weise kann man die Condition-Code-Flags NZCV (Bedingungsbits) des Prozessorstatuswort (CPSR) setzen?
  
  
  
  
  
  
  
  
  
  
- b) Wie wird die Pseudoinstruktion "ADR R1, Marke" vom Assembler umgesetzt? Schreiben Sie hierzu den Befehl in einen der vorgegebenen Programmrahmen und schauen Sie ihn sich im Debugger in der Mixed-Darstellung an. Vollziehen Sie die Umsetzung des Compiler nach und informieren Sie sich auch über Pipelining.
  
  
  
  
  
  
  
  
  
  
- c) Das Prozessorstatuswort hat den Wert 0x8000013, wenn der Befehl "BNE Marke" ausgeführt wird. Würde dann der Sprung an die (symbolische) Adresse Marke ausgeführt? Weisen Sie Ihre Antwort mit einem Programm nach.

**Aufgabe 3:**

Es ist ein Programm zu entwickeln, welches alle Werte einer Liste1 nach Liste2 kopiert. In den Listen steht an erster Stelle die Anzahl der Elemente der jeweiligen Liste. Liste1 ist, bis auf die erste Zahl (Anzahl der Elemente max. 255) eine Liste mit 8Bit großen vorzeichenbehafteten Zahlen (-128 bis +127). In Liste2 sollen die Zahlen aus Liste1, ausser die Anzahl der Elemente (die bleibt vorzeichenlos), als 32Bit große vorzeichenbehaftete Zahlen abgelegt werden.

**Aufgabe 4:**

Nach dem Kopiervorgang soll in einem weiteren Schritt die Liste2 aufsteigend sortiert werden. Hierzu erweitern Sie Ihr Programm von Aufgabe 3. Es gibt verschiedene Sortieralgorithmen (z.B. Bubblesort). Denken Sie daran, dass die Länge der Liste an erster Stelle unverändert stehen bleiben muss.

**Aufgabe 5:**

Dokumentieren Sie die Tests die gemacht wurden, um eine fehlerfreie Funktionalität der Programme nach zu weisen.

**Bericht**

Der erforderliche Praktikumsbericht dient zu Ihrer Nachbereitung des Praktikums und wird stichprobenhaft überprüft. Er hat auch den den zeilenweisen kommentierten Quelltext zu beinhalten. Haben Sie Ihre Berichte zu den Praktikumsterminen dabei.

```
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:

        .file      "aufgabe1.S"
        .text      @ legt eine Textsection fuer ProgrammCode + Konstanten an
        .align    2      @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren Adresse liegen
                        @ unteren 2 Bit sind 0
        .global   main   @ nimmt das Symbol main in die globale Sysmboltabelle auf
        .type    main,function

main:
    mov    r0, #0
    str    r1, [r0], #4      // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    eor    r0, r0, r0
    str    r1, [r0, #4]      // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    mov    r0, #0
    str    r1, [r0]!        // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    sub    r0, r0, r0
    str    r1, [r0, #4]!    // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    and    r0, r0, #0
    strb   r1, [r0, #1]!    // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    mov    r1, #4
    strb   r1, [r0, r1]! // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x_____
    bx     lr

.Lfe1:
        .size    main,.Lfe1-main

// End of File
```

\*\*\*\*\*

```
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:
//

        .file      "aufgabe2.S"
        .text      @ legt eine Textsection fuer ProgrammCode + Konstanten an
        .align    2      @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren Adresse liegen
                        @ unteren 2 Bit sind 0
        .global   main   @ nimmt das Symbol main in die globale Sysmboltabelle auf
        .type    main,function

main:

        bx     lr

.Lfe1:
        .size    main,.Lfe1-main

// End of File
```

\*\*\*\*\*

```
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:

        .file      "aufgabe3.S"
        .text
        .align    2          @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren Adresse liegen
                                @ unteren 2 Bit sind 0
        .global   main
                                @ nimmt das Symbol main in die globale Sysmboltabelle auf
        .type    main,function
main:
        push     {r4, r5, lr}      @ Ruecksprungadresse und Register sichern
```

kopieren:  
@ hier Ihr Programm zum Kopieren einer Byte-Tabelle (je 8Bit) in eine Word-Tabelle (je 32Bit) einfüegen

...

vorzeichen:  
@ hier Ihr Programm um die 8Bit-Zahlen auf vorzeichenrichtige 32Bit-Zahlen zu wandeln

...

sortieren:  
@ hier Ihr Programm um die vorzeichenrichtige Zahlen in Liste2 zu sortieren

...

```
fertig:
        pop      {r4, r5, pc}          @ Ruecksprungadresse und Register
TAB2:   .word   Liste2                @ Beispiel um an Adressen aus anderen Segmenten zu kommen
```

```
.Lfe1:
        .size   main,.Lfe1-main
```

```
// .data-Section für initialisierte Daten
        .data
```

```
// Erster Wert der Tabelle steht fuer die Anzahl der Werte in der Tabelle
Liste1: .byte   (Liste1Ende-Liste1), -9, 8, -7, 6, -5, 4, -3, 2, -1, 0, 127, 128
Liste1Ende:
```

```
// .comm-Section für nicht initialisierte Daten
        .comm  Liste2, (Liste1Ende-Liste1)*4 @ Speicherbereich mit der Groesse*4 von Liste1 reservieren
```

```
// End of File
```

\*\*\*\*\*