



**h\_da**

HOCHSCHULE DARMSTADT  
UNIVERSITY OF APPLIED SCIENCES

**fbi**  
FACHBEREICH INFORMATIK

## RECHNERARCHITEKTUR

WS2022

Termin 4

LOAD, STORE, bedingte Befehle, Speicherbereiche, ASCII-Tabelle

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]	$R1 := \text{mem}_{32}[R2]$
LDREQ R1, [R2]	$R1 := \text{mem}_{32}[R2]$ , falls das Z-Bit im Prozessorstatuswort gesetzt ist
LDRB R1, [R2]	$R1 := \text{mem}_8[R2]$
STR R1, [R2]	$\text{mem}_{32}[R2] := R1$
STRB R1, [R2]	$\text{mem}_8[R2] := R1$
ADR R1, Marke	$R1 := \text{PC} + (\text{Offset zur Marke})$
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	$R1 := \text{mem}_{32}[\text{PC} + (\text{Offset zur Hilfsmarke})]$ , dies ist eine Pseudoinstruktion

**Aufgabe 1:**

Auf welchen Adressen wird der Inhalt von Register r1 gespeichert? Ergänzen Sie die 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_____
ands r0, r0, #0
strb r1, [r0, #2]! // 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.  
Was passiert wenn die Marke sich nicht in der Sektion .text befindet?  
Wie kommen wir dann an die Adresse der Marke?
  
  
  
  
  
  
  
  
  
  
- c) Das Prozessorstatuswort hat den Wert 0x30000013, wenn der Befehl "SUBEQS R1, R1, R1" ausgeführt wird. Was steht danach im Register R1 und im CPSR? Weisen Sie Ihre Antwort nach.

**Aufgabe 3:**

Es ist ein Programm “kopieren” zu entwickeln, welches eine Zeichenkette von StringA nach StringB kopiert. Die Zeichenkette hat als Ende-Kennung ein Nullzeichen, ist also nullterminiert. Die zu verwendeten Strings beinhalten maximal 255 Zeichen.

**Aufgabe 4:**

Nach dem Kopiervorgang soll der StringB mit einem Programm “reduzieren” auf die im String vorhandenen Zahlen/Werte reduziert werden. Die Zahlen/Werte können sich in einem Wertebereich von 0 bis 255 bewegen. Die gefundenen Zahlen/Werte sind durch ein Semikolon von einander zu trennen.

**Zusatzaufgabe:**

In einem weiteren Programm “addieren” soll die Summe aller gefundenen Zahlen berechnet werden.

**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 zeilenweisen kommentierten Quelltext zu beinhalten. Haben Sie Ihre Ergebnisse und 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:
    adr    r1, marke

    bx    lr
marke:
    .word 0x12345678

.Lfe1:
    .size main,.Lfe1-main

// .data-Section fuer initialisierte Daten
    .data
marke1:
    .word 0x87654321

// End of File
```

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

```
.file "aufgabe3.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:
push {lr}        @ Ruecksprungadresse und evtl. weitere Register sichern
..
bl    kopieren
..
bl    reduzieren
..
bl    addieren
..

@ im Register R0 sollte die Summe aller ermittelten Zahlen/Werte stehen
pop {pc}

kopieren:
@ hier Ihr Programm zum Kopieren eines String
...
...

reduzieren:
@ hier Ihr Programm um alle Zahlen/Werte aus einen String zu filtern
...
...

addieren:
@ hier Ihr Programm zur Bildung der Summe aller gefundenen Werte
...
```



...

```
Adr_StringA: .word StringA      @ Hilfsvariable um an Adressen aus anderen Segmenten zu kommen
Adr_StringB: .word StringB      @ Hilfsvariable um an Adressen aus anderen Segmenten zu kommen
```

```
.Lfe1:
    .size main,.Lfe1-main      @ Programmgroesse berechnen
```

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

```
.data
```

```
// Liste von Zeichen
```

```
StringA: .asciz „Wertetabelle: 25, 11, 1, 263, 33 “
```

```
// .comm-Section fuer nicht initialisierte Daten
```

```
.comm StringB, 256 @ Speicherbereich fuer Arbeitsstring
```

```
.comm StringC, 256 @ Speicherbereich fuer gefundene Zahlen/Werte
```

```
// End of File
```