h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS2022

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 Beipiel der folgenden Befehle des ARM-Prozessors ein:

Instruktion	Bedeutung
ADDNE R1, R2, #1	R1 := R2 + 1, falls das Z-Bit im Prozessorstatuswort **nicht** gesetzt ist
LDR R1, [R2]	R1 := mem_{32}[R2]
LDREQ R1, [R2]	R1 := mem_{32}[R2], falls das Z-Bit im Prozessorstatuswort gesetzt ist
LDRB R1, [R2]	R1 := mem_{8}[R2]
STR R1, [R2]	mem_{32}[R2] := R1
STRB R1, [R2]	mem_{8}[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 **nicht** gesetzt ist
LDR R1, = Marke	R1 := mem32[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 4  danach steht in r0 0x 0
eor   r0, r0, r0
str   r1, [r0], #4    // Inhalt von r1 auf Adresse 0x 0  danach steht in r0 0x 4
mov   r0, #0
str   r1, [r0]!       // Inhalt von r1 auf Adresse 0x 0  danach steht in r0 0x 0
sub   r0, r0, r0
str   r1, [r0, #4]!   // Inhalt von r1 auf Adresse 0x 4  danach steht in r0 0x 4
ands  r0, r0, #0
strb  r1, [r0, #2]!   // Inhalt von r1 auf Adresse 0x 2  danach steht in r0 0x 2
mov   r1, #4
strb  r1, [r0, r1]!   // Inhalt von r1 auf Adresse 0x 6  danach steht in r0 0x 6
```

Aufgabe 2:

Bearbeiten Sie schriftlich die Fragen.

a) Auf welche Weise kann man die Condition-Code-Flags NZCV (Bedingungsbits) des Prozessorstatuswort (CPSR) setzen?

An eine arithmetische/logische Operation ein "s" anhängen, z.B. add -> adds, and -> ands, lsl -> lsls

logische Operationen, z.B. compare (cmp)

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?

add r1, pc, #0

Bei einem Code von:

```
        adr   r1, marke
        bx    lr
marke:  .word 0x12345678
```

Wenn die Adresse nicht in der Sektion .text ist, können wir mit dem Befehl LDR, also in unserem Beispiel LDR R1, =marke, auf die Adresse der Marke kommen

In das Register r1 wird die Adresse von "Marke", abhängig von der aktuellen Adresse des Program-Counters geschrieben. In unserem Beispiel (s.O.) wird in r1 pc + 0 gespeichert, da zu dem Zeitpunkt, wenn der Befehl ausgeführt wird, der PC bereits 2 Befehle weiter unten ist (Pipeline fetch - decode - execute).

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.

Der Befehl wird nur ausgeführt, wenn das Zero-Bit gesetzt wurde. Da in unserem Beispiel jedoch nur Carry und Overflow gesetzt sind, wird der Befehl nicht ausgeführt.
CPSR und R1 werden nicht verändert.

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 Zeichen die keine Buchstaben (a..z; A..Z) sind reduziert werden.

Aufgabe 5:

In einem weiteren Programm “sortieren” sollen die verbleibenden Zeichen im StringB aufsteigend sortiert werden.

Aufgabe 6:

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      sortieren
        pop     {pc}

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

reduzieren:
@ hier Ihr Programm  um einen String auf nur Buchstaben zu reduzieren
        ...
        ...

sortieren:
@ hier Ihr Programm um  einen String zu sortieren
        …
        …

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  „Dies ist eine Email-Adresse: Max_Mustermann@h-da.de !“

// .comm-Section fuer nicht initialisierte Daten
        .comm StringB, 256   @ Speicherbereich fuer zu sortierenden StringB
        .comm StringC, 256   @ Speicherbereich fuer zu sortierenden StringC

// End of File
```