h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS2025

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 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 := mem_{32}[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 einen Text nach der sogenannten **Caesar-Verschlüsselung** (auch als Cäsar-Chiffre, Cäsar-Algorithmus, Caesar-Verschiebung, Verschiebechiffre oder als Einfacher Caesar bezeichnet) codiert. Es ist ein einfaches symmetrisches Verschlüsselungsverfahren, das auf der monographischen und monoalphabetischen Substitution basiert. Der Einfachheit halber werden nur die 26 Buchstaben des lateinischen Alphabets ohne Unterscheidung von Groß- und Kleinbuchstaben als Alphabet für Klartext und Geheimtext verwendet und Sonderzeichen, Satzzeichen usw. nicht beachtet.

Das Verfahren:

Bei der Verschlüsselung wird jeder Buchstabe des Klartexts auf einen Geheimtextbuchstaben abgebildet. Diese Abbildung ergibt sich, indem man die Zeichen eines geordneten Alphabets um eine bestimmte Anzahl zyklisch nach rechts verschiebt (rotiert); zyklisch bedeutet, dass man beim Verschieben über Z hinaus wieder bei A anfangend weiterzählt. Die Anzahl der verschobenen Zeichen bildet den Schlüssel, der für die gesamte Verschlüsselung unverändert bleibt. Beispiel für eine Verschiebung um drei Zeichen:

```
Klar:    a b c d e f g h i j k l m n o p q r s t u v w x y z
Klar:    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Geheim:  D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Aus dem Klartext „caesar“ wird somit der Geheimtext „FDHVDU“. Das gleiche gilt für “cAeSaR”. Für die Entschlüsselung wird das Alphabet um dieselbe Anzahl Zeichen nach links rotiert.

Schauen Sie sich dazu die ASCII Tabelle an. Sie kodiert Zeichen in 7 Bit. So hat z.B. der Buchstabe ‘a’ den Code 97. Beachten Sie zudem, dass der Schlüssel eine beliebige vorzeichenlose Zahl sein kann, also auch größer als die Länge des Alphabets 26.

Schreiben Sie ihr Programm so, dass Sprünge möglichst vermieden werden, so dass die Befehlspipeline nicht unterbrochen wird.

Aufgabe 4:

Der verschlüsselte Text soll danach wieder entschlüsselt werden. Der Code soll in der Datei von Aufgabe 3 hinzugefügt 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 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

*****************************************************************************************
```

```
// Loesung Aufgabe 3 und 4 von Termin4 Rechnerarchitektur SS2024
// Variante Caesar Verschlüsselung
// Name:          Matrikelnummer:
// Name:          Matrikelnummer:
// Datum:
        .file "aufgabe3.S"
        .text               @ legt eine Textsection fuer PrgrammCode + 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

// ASCII Tabelle 65 - 90 : 'A' - 'Z'
// ASCII Tabelle 97 - 122: 'a' - 'z'


main:
        push    {r4, r5,r6,  lr} @ Ruecksprungadresse und Register sichern
        mov     r4, #3      @ Schlüssellänge in R4
        mov     r5, #29     @ andere Schlüssellänge in R5
        mov     r6, #55     @ andere Schlüssellänge in R6

@ hier Ihr Programm zum Verschlüsseln von Text  .....

        MOV     R0, R4              // Testfall auswählen, Schlüssel in R0
        // .....

@ hier Ihr Programm zum Rueckentschlüsseln vom verschlüsselten Text  ....
        // .....


ende:
        pop             {r4, r5, r6, pc} @ Ruecksprungadresse und Register

Text1: .word Text
Text2: .word  Text_verschluesselt @ Beispiel um an Adressen aus anderen Segmenten zu kommen
Text3: .word  Text_unverschluesselt
.Lfe1:
        .size main,.Lfe1-main

// .data-Section fuer initialisierte Daten
        .data

// Text, der zu verschluesseln ist
Text:   .asciz "CAESAR"
TextEnde:

// .comm-Section fuer nicht initialisierte Daten
        .comm Text_verschluesselt, (TextEnde-Text)        @ Speicherbereich mit der von Text reservieren
        .comm Text_unverschluesselt, (TextEnde-Text)      @ Speicherbereich mit der von Text reservieren

// End of File
```