RECHNERARCHITEKTUR

ab WS2021

Termin 4

LOAD, STORE, bedingte Befehle, Speicherbereiche

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.

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__0__ danach steht in r0 0x4
eor     r0, r0, r0
str     r1, [r0, #4]      // Inhalt von r1 auf Adresse 0x_____ danach steht in r0 0x0
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?

Rechenopertationen mit S

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.

c) Das Prozessorstatuswort hat den Wert 0x8000013, wenn der Befehl "BEQ 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 eines Vektor1 nach Vektor2 kopiert. In Vektor1 steht an erster Stelle die Anzahl der Elemente des Vektors. Vektor1 ist, bis auf den ersten Wert (Anzahl der Elemente max. 255) ein Vektor mit 8Bit großen vorzeichenbehafteten Werten (-128 bis +127). In Vektor2 sollen die Werte aus Vektor1, außer die Anzahl der Elemente (die bleibt vorzeichenlos), als 32Bit große vorzeichenbehaftete Werte abgelegt werden.

Aufgabe 4:

Nach dem Kopiervorgang soll in einem weiteren Schritt Vektor2 aufsteigend sortiert werden. Hierzu erweitern Sie Ihr Programm von Aufgabe 3. Es gibt verschiedene Sortieralgorithmen (z.B. Bubblesort). Denken Sie daran, dass die Länge des Vektors an erster Stelle unverändert stehen bleiben muss.

```
        .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, die 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
.Lfe1:  strb    r1, [r0, r1]!       // Inhalt von r1 auf Adresse 0x____ danach steht in r0 0x __
        bx      lr
        .size   main,.Lfe1-main
// End of File
*******************************************************************************************

//
        .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
*******************************************************************************************
```

```
        .file   "aufgabe3.S"
        .text                   @ legt eine Textsection fuer ProgrammCode + Konstanten an
        .align                  @ 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
main:   .type   main,function
        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) einfuegen
@  8Bit-Zahlen dabei auf vorzeichenrichtige 32Bit-Zahlen wandeln
        ...
        ...
        ...
sortieren:
@ hier Ihr Programm, um die vorzeichenrichtigen Zahlen in Liste2 zu sortieren
        ...
        ...

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

Adr_Liste1:     .word   Liste1              @ Hilfsvariable um an Adressen aus anderen Segmenten zu kommen
.Lfe1:
        .size   main,.Lfe1-main

// .data-Section für initialisierte Daten
        .data

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

// .comm-Section für nicht initialisierte Daten
        .comm   Liste2, (256*4) @ Speicherbereich fuer max. Groesse der Liste1 * 4

// End of File
```

****