

Rechnerarchitektur

Termin 4

LOAD, STORE, bedingte Befehle,
Speicherbereiche

RECHNERARCHITEKTUR

Termin 4

LOAD, STORE, bedingte Befehle, Speicherbereiche

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 Beispiel 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]	$R1 := \text{mem}_{32}[R2]$
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	$R1 := PC + (\text{Offset zur Marke})$
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 nicht gesetzt ist
LDR R1, = Marke	$R1 := \text{mem}_{32}[PC + (\text{Offset zur Hilfsmarke})]$, 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___
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.

- 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.

Termin 4

LOAD, STORE, bedingte Befehle, Speicherbereiche

Zu Aufgabe 1:

```
.global _start
_start:
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

_end:
*****
```

Zu Aufgabe 2:

```
.global _start
_start:

main:

    bx      lr

_end:
*****
```

Termin 4

LOAD, STORE, bedingte Befehle, Speicherbereiche

Zu Aufgabe 3 und 4:

```
.global _start
_start:
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) einfügen
@ 8Bit-Zahlen dabei auf vorzeichenrichtige 32Bit-Zahlen wandeln

```
...
...
...
```

sortieren:

@ hier Ihr Programm, um die vorzeichenrichtigen Zahlen in Liste2 zu sortieren

```
...
...
```

fertig:

```
ldmfd  sp!, {r4, r5, pc} @ Ruecksprungadresse und Register
```

```
Vektor1:      .byte (Vektor1Ende-Vektor1), -9, 8, -7, 6, -5, 4, -3, 2, -1, 0, 127, 128
```

```
Vektor1Ende:
```

```
TAB2:  .word  Vektor2
```

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

```
.data
```

```
.align 3
```

```
// Erster Wert der Tabelle steht fuer die Anzahl (max. 64) der Werte der Tabelle
```

```
Vektor2:      .space ((Vektor1Ende-Vektor1)*4)      @ Speicherbereich mit der Groesse*4 von Vektor1
reservieren
```

```
_end:
```

```
*****
```