h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS17

Termin4

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 folgende 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]
STR R1, [R2, #Offset]	mem_{32}[R2 + Offset] := R1
STR R1, [R2, R3]	mem_{32}[R2 + R3] := R1
STR R1, [R2, -R3]	mem_{32}[R2 - R3] := R1
STR R1, [R2, R3, LSL #2]	mem_{32}[R2 + R3 << 2] := R1
LDREQ R1, [R2]	R1 := mem_{32}[R2], falls das Z-Bit im Prozessorstatuswort gesetzt ist
LDRH R1, [R2]	R1 := mem_{16}[R2]
LDRB R1, [R2]	R1 := mem_{8}[R2]
STR R1, [R2]	mem_{32}[R2] := R1
STRH R1, [R2]	mem_{16}[R2] := R1
STRB R1, [R2]	mem_{8}[R2] := R1
LDRSB R1, [R2]	R1 := mem_{8}[R2] sign extended
ADR R1, Marke	R1: = PC+(Offset zur Marke) (Pseudobefehl!, nur gültig wenn Marke in der Nähe von PC ist)
LDR R1, =Marke	R1 := Adresse von Marke (Pseudobefehl!, gilt über gesamten Adressbereich)
LDR R1, =#Wert	R1 := #Wert (Pseudobefehl!, gilt auch für 32 Bit Werte)
B Marke	PC wird auf Adresse der Marke gesetzt, wenn Marke im Bereich +/-4092 vom PC entfernt ist, sonst Fehlermeldung
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

Aufgabe 1 (Vorbereitung):

Auf welchen Adressen wird der Inhalt von Register r1 gespeichert? Ergänzen Sie sie Kommentarzeilen.

```
main:
      mvn   r1, #222  @ Testzahl, genuegend gross und negativ

      ldr   r2, =Buffer
      add   r2, #0x20
      mov   r0, r2
      str   r1, [r0], #4       @ auf Adresse 0x_____ danach steht in r0 0x_____
      add   r1, #1
      mov   r0, r2
      str   r1, [r0, #-4]      @ auf Adresse 0x_____ danach steht in r0 0x_____
      add   r1, #1
      mov   r0, r2
      str   r1, [r0]!          @ auf Adresse 0x_____ danach steht in r0 0x_____
      add   r1, #1
      mov   r0, r2
      str   r1, [r0, #4]!      @ auf Adresse 0x_____ danach steht in r0 0x_____
      mvn   r1, r1
      mov   r0, r2
      strb  r1, [r0, #8]!      @ auf Adresse 0x_____ danach steht in r0 0x_____
      add   r1, #1
      strb  r1, [r0, #1]
      mov   r1, #0
      ldrsb r1, [r0]           @ Byte mit Vorzeichen laden
      ldrh  r1, [r0]
      mvn   r1, r1
      mov   r0, r2
      strh  r1, [r0, #12]      @ was steht auf Adresse r0+12 ? ______
      mov   r1, #4
      strb  r1, [r0, r1]!      @ auf Adresse 0x_____ danach steht in r0 0x_____
      strb  r1, [r0, -r1]!     @ auf Adresse 0x_____ danach steht in r0 0x_____
      str   r2, [r0, r1, lsl #3] @ auf Adresse 0x_____ danach steht in r0 0x_____
      bx    lr

      .data
      .align 2
Buffer: .space 100*4, 0
      @ End of File
.end
```

Aufgabe 2:

Es ist ein Programm tablecopy zu entwickeln, welches alle Werte einer Liste1 nach Liste2 kopiert. Das Programm bekommt als Parameter die Adressen von Liste2, Liste1 und N(Anzahl der zu kopierenden Elemente). Liste2 ist dabei ein uninitialisiertes Integerarray, Liste1 ein Array von signed Characters. Lesen sie die **signed Bytes** und speichern sie diese Werte als **Integer** in Liste2.

Als Hilfe gibt es ein C Programm, das einen Testrahmen zur Verfügung stellt und die Werte von Liste2 nach Programmausführung ausdruckt. Beachten sie die Übgergabe Konventionen der APCS. (Parameter 1-4 in R0..R3)

Aufgabe 3:

Nach dem Kopiervorgang soll in einem weiteren Schritt die Liste2 aufsteigend sortiert werden. Hierzu schreiben sie ein Programm bubblesort. Nutzen sie den Algorithmus von Bubblesort in pointer oder Arrayversion. Schreiben sie Bubblesort als Unterprogramm, das die Startadresse des Arrays und die Größe n übergeben bekommt. Auch hier steht Ihnen das Testprogramm bubbletest.c zur Verfügung. Die Parameter stehen wie üblich in den Registern R0 und R1.

Beachten sie, dass sie die Register R4-R11 nur benutzen dürfen, wenn sie die Werte dieser Register vorher auf dem Stack mit dem push Befehl gesichert haben. Es empfiehlt sich das Linkregister ebenfalls auf den Stack zu schieben.

Am Ende des Programms sind die Werte aller Register (außer LR) wieder mit einem pop Befehl zu restaurieren. Der Wert des Linkregisters kann dabei direkt in den Programcounter kopiert werden.

```
/* Dies ist die Pointer Version */

bubbleSort (int A[], int n)
{
	int *pA;
	int temp;
	do {
		swapped = false;
		n = n-1;
		pA = A;
		for (i=n; i > 0; i--) {
			if (*pA > *(pA+1) {
					temp = *pA;
					*pA = *(pA+1)
					*(pA+1) = temp;
					swapped = true;
				}
			pA++;
		}
	} while (swapped == true);
}
```

Bedenken sie, dass in diesem Algorithmus Pointer+1 einem Adresse+4 bei einer Wortgröße von 4 Byte entspricht

```
/* und jetzt die Array Version */

bubbleSort (int A[], int n)
{
	int temp;
	int i;
	do {
		swapped = false;
		n = n-1;
		for (i=0; i < n; i++) {
			if ( A[i] > A[i+1] ) {
					temp = A[i];
					A[i] = A[i+1];
					A[i+1] = temp;
					swapped = true;
				}
		}
	} while (swapped == true);
}
```

Bericht

Der erforderliche Praktikumsbericht dient zu Ihrer Nachbereitung des Praktikums und wird stichprobenhaft überprüft. Er beinhaltet auch den zeilenweisen kommentierten Quelltext. Haben Sie Ihre Berichte zu den Praktikumsterminen dabei.

```
// Zum Testen der in Assembler zu schreibenden Funktionen „tablecopy“ und „bubblesort“
// von Gerhard Raffius
// vom 28.3.2016
//

#include <stdio.h>

void tablecopy(int*, char*, int);
void bubblesort(int*, int);

char Table1[] = {9, -8, -7, 6, -5, 4, -3, 2, -1, 0, 127, 128};
int   Table2[sizeof(Table1)];

void printtable(int* table, int n)
{
      while(n--) printf("%d ", *table++);
      printf("\n");
}

int main (void)
{
      int i;
      tablecopy( Table2, Table1, sizeof(Table1) );
      printtable( Table2, sizeof(Table2)/sizeof(int));
      bubblesort( Table2, sizeof(Table2)/sizeof(int));
      printtable( Table2, sizeof(Table2)/sizeof(int));

      tablecopy( Table2, Table1, sizeof(Table1) );
      printtable( Table2, 2);
      bubblesort( Table2, 2);
      printtable( Table2, 2);

      tablecopy( Table2, Table1, sizeof(Table1) );
      printtable( Table2, 1);
      bubblesort( Table2, 1);
      printtable( Table2, 1);

      tablecopy( Table2, Table1, sizeof(Table1) );
      printtable( Table2, sizeof(Table1));
      bubblesort( Table2, 0);
      printtable( Table2, sizeof(Table1));
      return 0;
}
```