h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS2020

Termin 5

ARM: Stack, Unterprogramme, Sektionen, APCS

Name, Vorname	Matrikelnummer	Anmerkungen
Datum	Raster (z.B. Mi3x)	Testat/Datum

Legende: V:Vorbereitung, D: Durchführung, P: Protokoll/Dokumentation, T: Testat

Ziele:
Verständnis für STACK-Befehle und deren Nutzung bei Unterprogrammen. Ziel ist es Programme mit möglichst geringer Codegröße zu entwickeln, sowie den Umgang mit einem Debugger/Simulator zu festigen.

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 und in den ARM Procedure Call Standard (APCS) ein:

Instruktion	**Bedeutung**
STMDB R13!, {R4-R8, LR} STMFD R13!, {R4-R8, LR}	Speichert die Registerwerte R4 bis R8 sowie LR (=R14) an die Adresse, die in R13 (=SP) steht als voll absteigender Stack
LDMIA R13!, {R4-R8, PC} LDMFD R13!, {R4-R8, PC} LDM SP!, {R4-R8, PC}	Lädt den Speicherinhalt von der Adresse, die in R13 (=SP) steht in Form eines voll absteigenden Stacks in die Register R4 bis R8 sowie PC (=R15)

Aufgabe 1:
Die in der obigen Tabelle gezeigten Befehle aus dem ARM-Befehlssatz können durch welche vorzuziehenden synonyme Befehle ersetzt werden?

_____

_____

Kontrollieren Sie Ihre Lösung mit dem angebotenen zugehörigen Programmgerüstbeispiel.

Aufgabe 2:
Zeigen sie wie mit nur zwei „load und store multiple register instructions“ eine Liste von 16Byte-Werten (entspricht vier 32bit großen Worte) im Speicher kopiert werden kann. Im Register R0 steht die Adresse der Quellenliste und im Register R1 steht die Adresse des Ziels.

_____

_____

Kontrollieren Sie Ihre Lösung mit dem angebotenen zugehörigen Programmgerüstbeispiel.

Aufgabe 3:
In folgenden Tabellen ist jeweils ein Speicherauszug gezeigt. Welche Werte stehen in den Registern nach Ausführung des Blocktransferbefehls? R9 = 0x8000

Inhalt	Adresse
11	0x8014
10	0x8010
9	0x800C
8	0x8008
7	0x8004
6	0x8000
5	0x7FFC
4	0x7FF8
3	0x7FF4
2	0x7FF0

LDMDB R9, {R1, R2, R6}	LDMIA R9!, {R1, R2, R6}
R1: _____	R1: _____
R2: _____	R2: _____
R6: _____	R6: _____
R9: _____	R9: _____

Kontrollieren Sie Ihre Lösung mit dem angebotenen zugehörigen Programmgerüstbeispiel. Achtung die Werte im Beispiel weichen von den Werten der Aufgabe ab.
Achten Sie bei der praktischen Kontrolle auf evtl. abweichende Ergebnisse (Breakpoint bei main). Diskutieren und dokumentieren Sie Ihre Ergebnisse/Erkenntnisse.

Aufgabe 4:
Schreiben Sie ein beliebiges, kleines Programm in ARM Assembler, das durch Unterprogramme strukturiert wird. Folgende Anforderungen werden an das Programm und die Unterprogramme gestellt:

Die APCS Konvention ist einzuhalten.

Das Hauptprogramm soll (mindestens) drei Unterprogramme (UPx) aufrufen.

- UP1 benutzt nur Scratchregister und stellt keine Blattroutine dar (ruft somit weitere Unterprogramme auf)
- UP2 benutzt nur Scratchregister und stellt eine Blattroutine dar (ruft somit keine weiteren Unterprogramme auf)
- UP3 benutzt Nicht-Scratchregister und stellt keine Blattroutine dar (ruft z.B. UP1 auf)

Aufgabe 5:
Testen Sie das gegebene Programm mainfibo.c mit den zugehörigen noch leeren in Assembler geschriebenen Funktionen. Dokumentieren Sie Ihre Tests und die gefundenen Fehler.

Machen sie sich mit der Fibonacci-Funktion vertraut. Die Fibonacci-Funktion ist folgendermaßen definiert

X	Fibonacci(X)
X = 0	0
X = 1	1
X >= 2	Fibonacci(X-2) + Fibonacci(X-1)

Aufgabe 6:
Entwickeln, schreiben und testen Sie wenigstens eine der Assemblerfunktionen int fiborec(int) und/oder int fiboiter(int). Beschreiben Sie Ihre Funktionen und Diskutieren Sie die jeweiligen Vor- und Nachteile.

Zusatzaufgabe:
Es steht ein Speicher von zusätzlichen 1024 Byte zur Verfügung. Optimieren Sie die in Assembler geschriebenen Funktionen durch Nutzung des zur Verfügung stehenden Speichers so, dass einmal ermittelte Ergebnisse wie in einem Cache gehalten werden.

Rufen Sie die optimierten Funktionen aus mainfibo.c öfter auf und beobachten und dokumentieren Sie Ihre Beobachtungen.

Bericht
Der erforderliche Praktikumsbericht dient zu Ihrer Nachbereitung des Praktikums und wird geprüft. Er beinhaltet auch den zeilenweise kommentierten Quelltext.

```
//
// Programmgeruestbeispiel zur Aufgabe1
//
//Name:
//Datum:

        .file    "Aufgabe1.S"
        .text
        .align  2
        .globalmain
        .type   main, function
main:
        stmdb           r13!, {r4-r8, lr}
        stmfd           r13!, {r4-r8, lr}
@ welcher Befehl  wird durch den Compiler/Assembler genutzt

        ldmia           r13!, {r4-r8, lr}
        ldmfd           r13!, {r4-r8, pc}
@ welcher Befehl  wird durch den Compiler/Assembler genutzt


.Lfe1:
        .size   main,.Lfe1-main
```

```
//
// Programmgeruestbeispiel zur Aufgabe2
//
//Name:
//Datum:

        .file   "aufgabe2.S"
        .text
        .align  2
        .globalmain
        .type   main, function
main:
        push    {lr}

        adr     r0, Quelle
        ldr     r1, =Ziel

                                @Denken Sie an die APCS



        pop     {pc}

.Lfe1:
        .size   main,.Lfe1-main

Quelle:
        .byte   1, 2, 3, 4, 5, 6, 7, 8, 8, 10, 11, 12, 13, 14, 15, 16

// unintialisierter zur Verfügung stehender Speicherbereich
        .comm Ziel, 256
```

```
//
// Programmgeruestbeispiel zur Aufgabe3
//
//Name:
//Datum:

        .file   "aufgabe3.S"
        .text
        .align  2
        .globalmain
        .type   main, function
main:
        push    {R6, R9, LR}

        adr     r9, main

        ldmdb R9, {R1, R2, R6}
        ldmia  R9!, {R1, R2, R6}

        pop     {R6, R9, PC}

.Lfe1:
        .size   main,.Lfe1-main
```

```
// Programmgeruestbeispiel zur Aufgabe4
//
//Name:
//Datum:

        .file   "aufgabe4.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
        .globalmain             @ nimmt das Symbol main in die globale Sysmboltabelle auf
        .type   main,function
main:
        push    {lr}

        bl      up1
        bl      up2
        bl      up3

        pop             {pc}

// UP1 benutzt nur Scratchregister und stellt keine Blattroutine dar (ruft somit weitere
// Unterprogramme auf)
up1:



// UP2 benutzt nur Scratchregister und stellt eine Blattroutine dar (ruft somit keine weiteren
// Unterprogramme auf)
up2:
```

```
//UP3 benutzt Nicht-Scratchregister und stellt keine Blattroutine dar
up3:



.Lfe1:
        .size   main,.Lfe1-main

// End of File
```

```
// C-Programm zum Testen der in Assembler zu schreibenden Funktionen

#include <stdio.h>

int fiborec(int);
int fiboiter(int);
int fiboreccache(int);
int fiboitercache(int);

int c_fiborec(int);
int c_fiboiter(int);

int main()
{
        int n;
        int bis = 10;
        for ( n = 0; n <= bis ; n++ )
        {
                printf("Fiborec %d: %d\n", n, fiborec(n));
        }
        for ( n= 0; n <= bis ; n++ )
        {
                printf("Fiboiter %d: %d\n", n, fiboiter(n));
        }
        for ( n= 0; n <= bis ; n++ )
        {
                printf("Fiboitercache %d: %d\n", n, fiboitercache(n));
        }
        for ( n= 0; n <= bis ; n++ )
        {
                printf("Fiboreccache %d: %d\n", n, fiboreccache(n));
```

```
        }
        return 0;
}

int c_fiborec(int n)
{
        if (n==0)
                return 0;
        if (n<=2)
                return 1;

        return c_fiborec(n-1) + c_fiborec(n-2);

}

int c_fiboiter(int n)
{
        int a=0;
        int b=1;
        int neu;
        int x;
        for (x=1; x<=n; x++)
        {
                neu = a+b;
                a = b;
                b = neu;
        }
        return a;

}
```

```
        .file   "fibo.S"
        .text                   @ legt eine Textsection fuer PrgrammCode + Konstanten an
        .align  2               @ sorgt dafuer, dass nachfolgende Anweisungen auf einer durch 4 teilbaren Adresse liegen
                                @ d.h. die unteren 2 Bit sind 0
        .globalfiborec              @ nimmt das Symbol fiborec in die globale Sysmboltabelle auf
        .type   fiborec,function
fiborec:
        stmfd  sp!, {r4, r5, lr}



        ldmfd  sp!, {r4, r5, pc}

.fiborec_end:
        .size   fiborec,.fiborec_end-fiborec

        .globalfiboiter             @ nimmt das Symbol fiboiter in die globale Sysmboltabelle auf
        .type   fiboiter,function
fiboiter:



        bx      lr

.fiboiter_end:
        .size   fiboiter,.fiboiter_end-fiboiter

        .globalfiboitercache        @ nimmt das Symbol fiboitercache in die globale Sysmboltabelle auf
        .type   fiboitercache,function
fiboitercache :
        push   {lr}
```

```
        pop    {pc}

.fiboitercache_end:
        .size   fiboitercache,.fiboitercache_end-fiboitercache


        .globalfiboreccache          @ nimmt das Symbol fibocache in die globale Sysmboltabelle auf
        .type   fiboreccache,function
fiboreccache:
        push   {r4, r5, lr}


fiboreccache_end:
        pop    {r4, r5, pc}

.Lfe3:
        .size   fiboreccache,.Lfe3-fiboreccache

.data
                        .align  2
fiboliste:      .space 256*4, 0

.end                    @ End of File
```