h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS17

Termin5

Stack, Unterprogramme, Sektionen

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

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

Ziele:

Verständnis für Unterprogrammtechniken, STACK Befehle und deren Nutzung bei Unterprogrammen. Verständnis was Rekursion ist, wie der Zusammenhang zwischen rekursiven und iterativen Programmen ist und wie ein Cache in der Software funktioniert

Arbeitsverzeichnis:

Kopieren Sie sich das Verzeichnis von der Webseite

https://www.fbi.h-da.de/organisation/personen/raffius-gerhard/rechnerarchitektur.html

, 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**
push {R4-R7, LR}	Speichert die Registerwerte R4 bis R7 sowie LR (=R14) an die Adresse, die in R13 (=SP) steht als voll absteigender Stack
pop {R4-R7, PC}	Lädt den Speicherinhalt von der Adresse, die in R13 (=SP) steht in Form eines voll absteigenden Stacks in die Register R4 bis R7 sowie PC (=R15)

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

x	**Fibonaccio(x)**
x = 0	0
x = 1	1
x >= 2	Fibonaccio(x-2) + Fibonaccio(x-1)

Aufgabe 1:

Schreiben Sie ein Programm in ARM Assembler, das die Fibonaccio Funktion rekursiv berechnet. Beachten sie die APCS Konvention. Dieses Programm muss Variablen auf dem Stack speichern und ist keine Blatt Routine

```
define fiborec(n):
      if n<=1:
            return n
      return fiborec(n-1)+fiborec(n-2)
```

Aufgabe 2:

Schreiben Sie ein Programm in ARM Assembler, das die Fibonaccio Funktion iterativ berechnet. Beachten sie die APCS Konvention. Dieses Programm kann man unter Benutzung von Scratch Registern (r0-r3, ip) und ohne Stackzugriff schreiben.

```
define fiboiter(n):
      if n<=1:
            return n
      a = 0
      b = 1
      for zaehler = 1..n:
            neu = a+b
            a = b
            b = neu
      return a
```

Welches der beiden Programme 1 und 2 wird für große x schneller berechnet und warum?

Aufgabe 3:

Schreiben sie das Programm als rekursives Programm mit Cache. Speichern sie die errechneten Werte in dem Array fiboliste

```
fiboliste = leer

define fibocache(n):
      if n<=1:
            return n
      if fiboliste[n] != 0 :
            return fiboliste[n]
      return fiboliste[n] = fibocache(n-1)+fibocache(n-2)
```

Bericht

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

Als Hilfestellung erhalten sie ein Main Programm, das die drei Fibonaccio Funktionen aufruft, einen Rahmen für die Unterprogramme und ein Makefile

mainfibo.c:

```
#include <stdio.h>

int fiborec(int n);
int fiboiter(int n);
int fibocache(int n);

int main()
{
	int n;
	for ( n= 0; n <= 30 ; n++ )
	{
		printf("Fiborec %d: %d\n", n, fiborec(n));
	}
	for ( n= 0; n <= 30 ; n++ )
	{
		printf("Fiboiter %d: %d\n", n, fiboiter(n));
	}
	for ( n= 0; n <= 30 ; n++ )
	{
		printf("Fibocache %d: %d\n", n, fibocache(n));
	}
	return 0;
}
```

makefile:

```
CC = arm-elf-gcc
CFLAGS = -O2 -g -c
LDFAGS = ""
ASFLAGS = -g -c
OBJS = mainfibo.o fibo.o
NAME = testfibo

$(NAME).elf: $(OBJS)
	$(CC) $(LDFLAGS) $(OBJS) -o $(NAME).elf

%.o: %.s
	$(CC) $(ASFLAGS) $< -o $@

%.o: %.S
	$(CC) $(ASFLAGS) $< -o $@

%.o: %.c
	$(CC) $(CFLAGS) $< -o $@
```

fibo.S

```
@ Name:                 Matrikelnummer:
@ Name:                 Matrikelnummer:
@ Datum:

        .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
@-----------------------------------------------------------------------------------------------
        .global  fiborec        @ nimmt das Symbol fiborec in die globale Symboltabelle auf
        .type    fiborec,function
fiborec:
        @  Code für fiborec
.Lfe1:
        .size    fiborec,.Lfe1-fiborec

@-----------------------------------------------------------------------------------------------
        .global  fiboiter       @ nimmt das Symbol fiboiter in die globale Symboltabelle auf
        .type    fiboiter,function
fiboiter:
        @  Code für fiboiter


.Lfe2:
        .size    fiboiter,.Lfe2-fiboiter

@-----------------------------------------------------------------------------------------------
        .global  fibocache      @ nimmt das Symbol fibocache in die globale Symboltabelle auf
        .type    fibocache,function
fibocache:
        @  Code für fibocache


.Lfe3:
        .size    fibocache,.Lfe3-fibocache

.data
                .align  2
fiboliste:      .space 100*4, 0

.end                            @ End of File
```