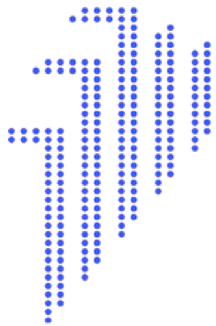


h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi
FACHBEREICH INFORMATIK

RECHNERARCHITEKTUR

SS2020

Termin 5

ARM: Stack, Unterprogramme, Hochsprache, 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 die Interaktion zwischen Hochsprache- und Assemblerprogramme durch das Einsetzen der APCS zu verinnerlichen.

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:

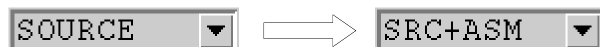
Instruktion	**Bedeutung**
STMDB R13!, {R4-R8, LR} STMFD R13!, {R4-R8, LR}	Speichert die Registerwerte R4 bis R8 sowie LR (=R14) an die Adresse, die in R13 (=SP) steht als voll absteigender Stack
LDMIA R13!, {R4-R8, PC} LDMFD R13!, {R4-R8, PC}	Lädt den Speicherinhalt von der Adresse, die in R13 (=SP)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

____

____

Zum Überprüfen Ihrer Antwort können Sie im Debuggerfenster die Ansicht auf „SRC+ASM“ oder „MIXED“ stellen.



Aufgabe 2:

Zeigen sie wie mit nur zwei „load und store multiple register instructions“ eine Liste von 16Byte-Werten im Speicher kopiert werden kann. Im Register R0 steht die Adresse der Quellenliste und im Register R1 steht die Adresse des Ziels.

____

____

Aufgabe 3:

Welche Register müssten Sie mit Werten befüllen, bevor Sie aus einem Assemblerprogramm die in der Hochsprache C geschriebene Funktion AdditionMitRest aufrufen? Tragen Sie im Assemblerprogramm die richtigen Register-Indizes ein.
Was steht am Ende des Assemblerprogramms im Register R0? Warum wird das Linkregister „gerettet“ und „wiederhergestellt“?

```
int AdditionMitRest(int a, int b, int mod)
{
   return (a+b) % mod;
}
```

```
main:
   PUSH  {LR}

   MOV   R_, #0x2    // a
   MOV   R_, #0x3    // b
   MOV   R_, #0x4    // c
   BL    AdditionMitRest

   TST       R0, R0
   MOVEQ   R0, #0x4
   MOVNE   R0, #0x5

   POP {PC}
```

Aufgabe 4:

Implementieren Sie eine Assembler-Funktion `fiborec`, welche die Fibonacci-Zahl der angegebenen Stelle in der Fibonacci-Reihe durch einen rekursiven Algorithmus berechnet.

Hierbei gilt:

```
fiborec(n):
   if n < 2:
      return n;

   return fiborec(n-1) + fiborec(n-2);
```

Die Funktion `fiborec` wird in der Datei `fibonacci.S` definiert und durch den Hochsprachencode in der Datei `aufgabe4.c` aufgerufen.

Machen Sie sich erst mit den Signaturen der Funktionen vertraut, indem Sie die Datei `testSuite.h` studieren.

Bedenken Sie, dass Sie sich unbedingt an die APCS halten müssen, damit die Interaktion des Hochsprachencodes und Ihrer Assembler-Implementierung auch funktioniert!

Überprüfen Sie das Laufzeitverhalten Ihrer Implementierung im Debugger: Ihre implementierte Funktion wird in der main-Funktion mehrmals aufgerufen und das Ergebnis der Berechnungen in einem Debug-Konsolenfenster angezeigt.

Dieser Vorgang könnte u.U. einige Sekunden dauern. Sie können gerne auch die Laufdauer der Tests abändern.

Stellen Sie sicher, dass die Konsolen-Ausgabe die Richtigkeit Ihres Algorithmus‘ bestätigt.

Aufgabe 5:

Implementieren Sie nun die Assembler-Funktion `fiboreccache`, welche wie in Aufgabe 4 die Fibonacci-Zahl für eine gegebene Stelle berechnet, jedoch für schon berechnete Werte das Ergebnis aus einem Zwischenspeicher abliest/ablegt.

Hierbei gilt:

```
fiboreccache(n):
   If n < 2:
      return n;

   if fibocache[n-2] == valid:
      return fibocache[n-2];

   fibocache[n-2] = fiboreccache(n-1) + fiboreccache(n-2);
   return fibocache[n-2];
```

Überlegen Sie sich, wie Sie die Validität eines Cache-Eintrags am geschicktesten überprüfen. Der Fibo-Cache wird mit 0-Bytes vorinitialisiert und befindet sich in der BSS-Sektion des Speichers (aus vorigen Praktika auch besser bekannt unter der COMM-Sektion). Der Cache kann bis zu 65536 mögliche Ergebnisse speichern.

Vergleichen Sie das Laufzeitverhalten von `fiboreccache` und `fiborec`. Welche der beiden Funktionen kann in der angegebenen Test-Zeit mehr Werte berechnen und warum?

Zusatzaufgabe 1:
Implementieren Sie die Assembler-Funktion `fiboiter`, welche die gewünschte Fibonacci-Zahl gemäß der vorigen Aufgaben durch einen **iterativen** Algorithmus berechnet. Schauen Sie sich hierzu Referenzimplementierungen im Internet an. Versuchen Sie – sofern möglich – nur Scratch-Register für die Implementierung zu nutzen, um die Funktion noch performanter zu machen!

Überprüfen Sie das Laufzeitverhalten der Funktion, indem Sie die entsprechende Zeile in der main-Funktion auskommentieren.

Zusatzaufgabe 2:
Implementieren Sie nun die Assembler-Funktion `fiboitercache`. Diese soll, wie in der vorigen Zusatzaufgabe, die Fibonacci-Zahlen auf iterativem Weg berechnen, jedoch schon berechnete Werte zwischenspeichern (s. Aufgabe 5).

Überprüfen Sie das Laufzeitverhalten der Funktion, indem Sie die entsprechende Zeile in der main-Funktion auskommentieren.
Hierbei wird der Cache vorher wieder zurückgesetzt und die `fiboitercache`-Funktion zweimal hintereinander ausgeführt.
Welche Beobachtung machen Sie bei den beiden Ausführungen der Funktion? Wie können Sie diesen Effekt erklären?

Welcher der 4 Implementierungen war nun am schnellsten (bzw. hat die meisten Berechungen innerhalb der angegebenen Zeit machen können)?

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