universität innsbruck

Proseminar Rechnerarchitektur

Einsendeaufgabe 2

Wintersemester 2021/22

13. Dezember 2021

Diese Einsendeaufgabe muss bis zum **12. Jänner 2021 um 23:59 Uhr** bearbeitet und im OLAT abgegeben werden. Die Einsendeaufgabe besteht aus den vier Teilaufgaben 1.1 bis 1.4. Ihre Abgabe fließt in die Endnote für das Proseminar Rechnerarchitektur mit ein.

Vorbereitung Laden Sie zunächst die Datei `arm-einsendeaufgabe.zip` aus dem OLAT herunter. Melden Sie sich am ZID-GPL an und erstellen Sie einen neuen Ordner. Kopieren Sie die ZIP-Datei in den neuen Ordner. Wechseln Sie in den neuen Ordner und entpacken Sie unsere Vorlage mit dem Befehl `unzip arm-einsendeaufgabe.zip`.

Jetzt sollte der Befehl `ls` die vier Order `max`, `functions`, `fold` und `fib` sowie ein `Makefile` auflisten. Jeder der Ordner entspricht einer Teilaufgabe. Wechseln Sie zum Bearbeiten einer Teilaufgabe in den entsprechenden Ordner. Darin können Sie Ihr Programm mit `make run` kompilieren und ausführen.

Tests zur Selbstkontrolle Wir haben zu jeder Teilaufgabe eine Reihe von Tests programmiert, die Sie mit `make test` ausführen können. Die Tests geben Ihnen direktes Feedback zum Status Ihrer Lösung. Aber Achtung: Zur Bewertung Ihrer Abgabe werden wir weitere automatische Tests hinzufügen und auch manuelle Kontrollen nutzen. Lesen Sie die Aufgabenstellung genau! Orientieren Sie sich nicht ausschließlich an den bereitgestellten Tests.

Abgabe Wechseln Sie in den Order mit der Datei `arm-einsendeaufgabe.zip`. Führen Sie `make abgabe.zip` aus. Sie sollten nun eine Datei `abgabe.zip` im Arbeitsverzeichnis vorfinden. Überprüfen Sie den Inhalt dieses Archivs (ZIP-Datei). Das Archiv sollte nur Ihre vier bearbeiteten ARM-Assembler-Dateien enthalten und sonst nichts. Laden Sie das Archiv zur Abgabe im OLAT hoch.

Bewertung Die Bewertung erfolgt teilweise automatisch. Stellen sie sicher, dass ihr Kode kompiliert. Halten Sie sich genau an die Vorgaben zum Bearbeiten der Aufgaben und zur Abgabe, ansonsten können wir keine fehlerfreie Bewertung garantieren. Plagiate (in Teilaufgaben) werden mit 0 Punkten (auf die gesamte Einsendeaufgabe) für alle Beteiligten geahndet. In besonders schweren Fällen und im Wiederholungsfall wird das gesamte Modul negativ bewertet. Zur Erinnerung: Das Austauschen von konkreten Lösungshinweisen ist *nicht* zulässig.

Kommentare Bitte kommentieren Sie Ihren Kode ausführlich. Das hat mehrere Vorteile:

- Wir können besser nachvollziehen, was Sie sich bei der Bearbeitung gedacht haben.
- Sie zeigen, dass Sie Ihr Programm verstehen und können damit Plagiatsvorwürfe abwehren.
- Eine versehentliche Erkennung als Plagiat wird noch unwahrscheinlicher.

1 Tut Max was es soll?

Wechseln Sie in den Ordner `max`. Kopieren Sie Ihre eigene Datei `max.S` von Zettel 10, Aufgabe „Mit maximaler Geschwindigkeit voraus“, in Ihren Arbeitsorder. Führen Sie unsere Tests aus (`make test`). Besteht Ihre `max`-Funktion alle Tests? Falls nicht, korrigieren Sie Ihre `max`-Funktion entsprechend.

- Ihre Funktion sollte mit beliebigen 32-bit Array-Größen und -Inhalten funktionieren.
- Ihre Funktion sollte die ARM-Aufrufkonventionen aus der Vorlesung beachten.
- Ihre Funktion sollte im Falle eines ungültigen Aufrufs (z.B. Länge Null) nicht das gesamte Programm zum Absturz bringen.

2 Kopieren, Einfügen, Anpassen

Als nächstes wollen wir zwei Funktionen in ARM implementieren, die der `max`-Funktion stark ähneln: `min` und `sum`.

Wechseln Sie in den Ordner `functions`. Kopieren Sie Ihre `max`-Funktion von Aufgabe 1 an die vorbereitete Stelle in der Datei `functions.S`.

Implementieren Sie die fehlenden Funktionen `min` und `sum` in der Datei `functions.S`. Die Funktion `min` soll das kleinste Element des übergebenen Arrays ermitteln und zurückgeben. Die Funktion `sum` soll die Summe aller Array-Elemente berechnen und zurückgeben.

Die drei Funktionen sollten folgende Anforderungen erfüllen.

- Ihre Funktionen sollten mit beliebigen 32-bit Array-Größen und -Inhalten funktionieren.
- Ihre Funktionen sollten die ARM-Aufrufkonventionen aus der Vorlesung beachten.
- Ihre Funktionen sollten im Falle eines ungültigen Aufrufs (z.B. Länge Null) nicht das gesamte Programm zum Absturz bringen.

3 Goldener Schnitt

Gegeben sei die folgende Definition der Fibonacci-Folge:

$$\begin{aligned}\text{fib}(0) &= 0\\ \text{fib}(1) &= 1\\ \text{fib}(n) &= \text{fib}(n-1) + \text{fib}(n-2)\end{aligned}$$

Ihre Aufgabe ist es, die Funktion `extern int fib(unsigned int n)` mit Rekursion in Assembler zu implementieren, sodass sie von einem C-Programm aus aufgerufen werden kann. Schreiben sie eine rekursive Funktion.

Gehen Sie dazu wie folgt vor.

a) Wechseln Sie in den Order `fib`.

b) Implementieren Sie die fehlende Funktion `fib` in der Datei `fib.S`.

c) Testen Sie Ihre Funktion mit `make run` und `make test`.

4 Falten, bitte!

Die Funktionen `max`, `min` und `sum` von Aufgabe 2 sind sehr ähnlich. Können Sie die Redundanz vermeiden?

Überlegen Sie sich zunächst, wie Sie die drei Funktionen mit `fold` in einer funktionalen Programmiersprache (z.B. Haskell) implementieren würden.

Wechseln Sie in den Ordner `fold`.

Implementieren Sie die fehlenden Funktionen `fold`, `agg_min` und `agg_sum` in der Datei `fold.S`. Ziel ist es, dass die Funktionen `max`, `min` und `sum` die Anforderungen aus Aufgabe 2 erfüllen. Testen Sie Ihre Funktionen mit `make test`.

Tipp: Wenn `r0` die Adresse einer Funktion `f` enthält, dann können Sie `f` wie folgt aufrufen.

```
1 MOV lr, pc
2 MOV pc, r0
```