Name der Vorlesung
Aufgaben zu Einführung in die ARM Assembler Sprache

1 Aufgabe 1 - Verständniss

1.1 Allgemeines

- Welches Speichermodell verwenden die ARM Architekturen ?
- Welche Wortbreite besitzt eine Instruktion ?
- Welche der ARM Instruktionen verwenden Erweiterungsworte ?
- Wie werden Instruktionen verarbeitet ?
- Angenommen zum Zeitpunkt n wird eine ADD Instruktion, an der Adresse `0xAFF6`, tatsächlich ausgeführt, an welcher Adresse befindet sich der `PC`, wenn sich der Prozessor im ARM Modus befindet?
- Was ist die Tool Chain? Geben pro Programmierwerkzeug eine kurze Beschreibung.
- Aus welchen Sektionen besteht eine Assembler Quelldatei?
- Zeichnen Sie das typische Speicher-Layout eines Prozesses unter Linux.

1.2 Bits and Bytes

- Welche Größe (in Bit) besitzen: `BYTE, HALF WORD, WORD, DOUBLE WORD`?
- Welche Endianness wird unter ARM genutzt?
- Beschreiben Sie `Little Endian` in einem Satz.

1.3 Register

- Was sind `Callee Save Register`?
- Welches Register (Rx) stellt den Program Counter dar?
- Was ist der Zweck des Link Registers? Was ist in dieser Hinsicht der Unterschied zu x86?
- Nennen Sie die Flags, die für bedingte Instruktionen zur verfügung stehen.
- Nennen Sie zwei Prozessor Modi und beschreiben Sie diese kurz.

2 Data Processing Instructions

Für die nachfolgenden Aufgaben wird ein Arm Computer, wie z.B. Raspberry Pi, oder ein Emulator, wie z.B. Quemu, benötigt, sowie ein Linux Betriebssystem.

Die Beschreibungen beziehen sich auf eine ARM Entwicklungsumgebung bestehend aus eine `Raspberry Pi 3` auf dem `Raspbian` installiert wurde.

2.1 Einrichten der Entwicklungsumgebung

Sie können den Raspberry Pi mittels HDMI und USB direkt mit einem Monitor, sowie Tastatur verbinden.

Als Alternative können Sie sich auch mittels `SSH`, von einem anderen Computer aus, mit dem Raspberry Pi verbinden. Dafür müssen Sie sich im selben Netzwerk befinden. Für den Verbindungsaufbau benötigen Sie weiterhin die IP-Adresse des Gerätes. Angenommen Sie möchten sich als Nutzer `pi` auf dem Gerät mit der IP-Adresse `192.168.62.255` einlogen, dann können Sie dies über die Konsole mit folgendem Befehl bewerkstelligen:

```
ssh pi@192.168.62.255
```

2.2 Dividieren mal anders

2.2.1 a)

Auf modernen Systemen ist ein Dividierwerk standard, mit dem ein Zahl durch eine andere, mittels Hardware, geteilt werden kann. Die Armv-8 Architektur bietet die Instruktionen `UDIV` und `SDIV` um vorzeichenlose bzw. vorzeichenbehaftete Zahlen zu dividieren. Ältere Architekturen besitzten diese Möglichkeit nicht. Dort können Divisionen in Software, d.h. durch eine Funktion, realisiert. Solche Dividieralgorithmen sind jedoch sehr ineffizient und benötigen viele Taktzyklen.

Berechnen Sie `(7 * 5) / 3 + 21` mit so wenigen Instruktionen wie möglich, ohne die Verwendung eines Multiplikations- oder Dividierbefehls und ohne die einzelnene Zahlen im Vorraus zusammenzufassen.

2.2.2 b)

Für einen Zyklischen Buffer der größe `buffer_size` der mittels `offset` indiziert wird, könnte eine erhöhung um `n` Wörter wie folgt aussehen:

```
offset = (offset + n) % buffer_size;
```

Angenommen der obige Ausdruck benötigt 50 Taktzyklen zur berechnung, geben Sie eine, ihrer Meinung nach, effizientere Lösung an. Implementieren Sie ihre Lösung in Assembly. Wie viele effektive Taktzyklen benötigt Ihre Implementierung?