Hochschule RheinMain
Fachbereich Design Informatik Medien
Marcus Thoss, M.Sc.

Mikroprozessortechnik
SS 2020
LV 2522

Übungsblatt 3

Bei den folgenden Übungen wird nach einem Beginn in C die Assembler-Programmierung von ARM-Prozessoren (statt Atmel/Microchip AVR) betrachtet.

Konkret soll auf einer mit dem Programm `qemu` realisierten *Emulation* eines Stellaris LM3S6965 mit ARM Cortex-M3-Core von Texas Instruments programmiert werden.

Greifen Sie für die ersten Schritte auf die Anmerkungen in der Vorlesung und das auf der Laborserver-Seite verlinkte User Guide und Technical References Manual des Cortex-M3 zurück. Versuchen Sie, sich zunächst auf wenige Instruktionen zu beschränken, die denen der AVR-Programmierung verwandt sind, also etwa `BEQ`, `LDR/STR/MOV` (= `LDS/STS/MOV` auf dem AVR) oder `ADD`. Beachten Sie die veränderte Syntax und Anzahl/Reihenfolge der Operanden!

Hinweise zur Installation:

Um Quellcode für ARM-Prozessoren auf eine beliebigen Linux-PC zur entwickeln, muss, ähnlich wie für AVR-Programmierung, eine weitere "Cross-Compiler"-Variante des `gcc` installiert werden. Um die unten angegebenen Tool-Aufrufe wie `arm-none-eabi-gcc` verfügbar zu haben, müssen Sie entsprechende Pakete auf Ihrem Entwicklungs-PC installieren. Bei mit Debian (also auch Ubunbtu) verwandten Distributionen ist dies das Paket `gcc-arm-none-eabi`. Weiterhin werden der Emulator `qemu`, ähnlich `simulavar`, für die Zielplattform ARM (Paket `qemu-system-arm`) und ein passender `gdb` (mit enthalten im Paket `gdb-multiarch`) benötigt.

Die genannten Tools stehen auch auf den ITS-Laborrechnern (`its01.local.cs.hs-rm.de`, `its02...` usw.) zur Verfügung. Da diese Rechner inzwischen wieder betriebsbereit sind, sollten Sie sie anstelle der PCs der anderen Linux-Pools (`lx1-01...`) verwenden, da dort nicht immer alle Spezialtools für unsere LVen installiert sein werden.

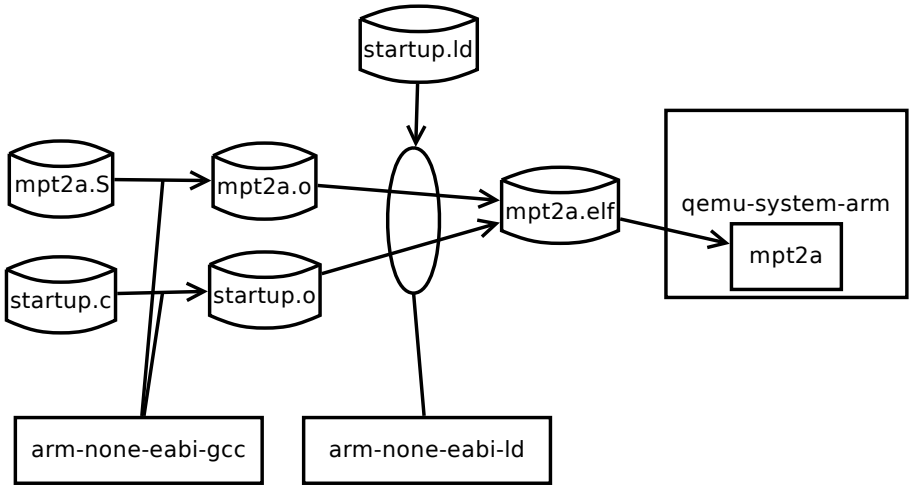
Aufgabe 3.1 (ARM-Assembler):

In diesem Schritt erzeugen und testen Sie das erste ARM-Programm.

- Kopieren Sie das minimalistische Assembler-Programm `mpt3a.S` vom Laborserver in ein neues Übungsverzeichnis.

- Um das Programm als einzigen Code auf dem ARM-Emulator laufen zu lassen (auf dem emulierten Prozessor läuft ja dann kein Betriebssystem!), müssen Sie eine Initialisierung des Prozessors nach dem "Einschalten" und initialen Reset durchführen. Hierzu finden Sie auf dem Laborserver die Dateien `startup.c`, `startup_ARMCM3.S` und `startup.ld`. Kopieren Sie diese in Ihr Projektverzeichnis.

- Für das Gesamtprogramm müssen `mpt3a.S` und `startup.c` compiliert und unter Verwendung des Linker-Skripts `startup.ld` zum Programm `mpt3a.elf` verlinkt werden:



  Hinweis: Wenn Sie an weiteren Details interessiert sind, können Sie sich später auch die umfangreichere Version des Startup-Codes `startup_ARMCM3.S` ansehen und diese statt `startup.c` compilieren und verwenden. In diesem Fall müssen Sie noch zusätzlich die `gcc`-Optionen `-D__NO_SYSTEM_INIT -D__START=main` angeben. Für den Beginn genügt aber `startup.c`.

- Verwenden Sie folgende Compiler- und Linker-Aufrufe für den Build-Vorgang:

```
arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb -g -c <Quelldatei>

arm-none-eabi-ld -g -T startup.ld <.o-Dateien> -o <.elf-Programmdatei>
```

- Zum Emulieren und Debuggen (in separaten Terminal-Fenstern) verwenden Sie:

```
qemu-system-arm -M lm3s6965evb --kernel <.elf-Programmdatei>
                --serial null -nographic -S -s

arm-none-eabi-gdb <.elf-Programmdatei>
```

  Anschließend können Sie sich, wie von Embedded-Targts oder `simulavr` gewohnt, aus dem `gdb` mit `target remote` mit der Emulation verbinden (kein `load` in diesem Fall). Der Standard-TCP-Port hierfür ist bei `qemu` Port 1234.

- Sehen Sie sich die Kommandos der Monitor-Kommandozeile von `qemu` mit `help` an und recherchieren Sie die Bedeutung der in der Aufgabe vorgegebenen Startoptionen (z.B. `-M` oder `-S`).

Hinweise:

In `qemu` können Sie den Emulator mit `system_reset` zurücksetzen ("Reset" des emulierten Systems). Anschließend ist allerdings auch wieder ein `target`-Kommando von `gdb` aus nötig, um Ihr Programm neu zu laden. Für beides müssen Sie `qemu` und `gdb` nicht schließen.

Verwenden Sie `layout regs`, um sich in `gdb` die Register anzusehen. Wie können Sie die Registerinhalte in `qemu` ausgeben? Nutzen Sie für das Ausprobieren auch die Tatsache, dass beide Tools Tab-Completion (2xTAB eingeben) unterstützen.

Sollte auf Ihrem Ubuntu-System `arm-none-eabi-gdb` nicht installierbar sein, installieren Sie stattdessen `gdb-multiarch`.

Aufgabe 3.2 (ggT auf ARM):

Portieren Sie Ihre ggT-Implementierung aus Aufgabe 1.3 in ARM-Assemblercode und testen Sie sie unter `qemu`.

Hinweise:

- Nutzen Sie Parameter-Übergabe in Registern. Halten Sie sich dabei an den "Procedure Call Standard for the ARM Architecture" (AAPCS).
- Beachten Sie, dass die Instruktion `BL` (Ersatz für AVR-`RCALL`) die Rücksprungadresse nicht auf dem Stack, sondern in Register `R14` (auch `LR` genannt) ablegt.

  Bei der Rekursion in der ggT-Lösung würde ein Sprung aus einer Subroutine in eine weitere, tiefer verschachtelte Ebene die letzte Rücksprungadresse in `LR` überschreiben! Sie müssen also bei Schachtelungen `LR` auf dem Stack sichern und vor dem Rücksprung wieder vom Stack holen.