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

Mikroprozessortechnik
SS 2020
LV 2522

Übungsblatt 5

Aufgabe 5.1 (C und Assembler kombiniert):

In dieser Aufgabe werden C-Funktionen in Assembler realisiert und die serielle UART-Schnittstelle des ARM Cortex-M3 verwendet. Die Funktionen dienen dem Zugriff auf die Instruktionen `ADD` bzw. `ADDS` aus C heraus, um in einem kleinen C-Demponstratorprogramm zeigen zu können, wie die Statusflags durch `ADDS` verändert werden.

Ihr Programm wird letztlich in etwa folgenden Ablauf unerstützen:

```
$ qemu-system-arm -M lm3s6965evb --kernel mpt5a.elf -nographic
input an eight-digit hex number:ffffffff
input another eight-digit hex number:ffffffff
Numbers are
11111111111111111111111111111111 and
11111111111111111111111111111111
Adding them yields these flags:
N-C--
and this result:
11111111111111111111111111111110
input an eight-digit hex number:
```

a) Beginnen Sie damit, sich vom LV-Server das Archiv `mpt5a_src.tar.gz` herunterzuladen (es enthält die Dateien `mpt5a.c`, `stellaris_uart.c`, `stellaris_uart.h`, `addfunc.S` `addfunc.h`, `startup.c` und `startup.ld`), in ein ein lokales Verzeichnis zu entpacken und ein Makefile zu schreiben, das aus diesen Quellen das ARM-Programm `mp5a.elf` erzeugt.

Sehen Sie dabei `gcc`-Flags für folgende Eigenschaften vor: Debugging und M3-Cortex-CPU mit Thumb-Befehlssatz.

Für den *Linker*schritt sind gegenüber den letzten Aufgaben diesmal noch folgende zusätzliche Flags *am Ende der Optionsliste* zum Hinzulinken der C-Standardbibliothek (hier für `strtoul()` benötigt) erforderlich:
`-L /usr/lib/arm-none-eabi/lib/thumb/v7-m/ -lc`

b) Sehen Sie sich die vorgegebenen Quellen mit ihren Kommentaren an und verschaffen Sie sich einen Überblick. Als ersten Test sollen Sie wieder eine "Hello, world!"-Ausgabe realisieren.

Die Funktion `int uartPutString(const char *s)` ist dafür gedacht, allerdings benötigt sie die (noch unvollständige) Implementierung von `void uartOut(unsigned char b)`. Sie wissen von Blatt 4 bereits, wie Sie ein Zeichen durch Schreiben des ASCII-Wertes in das Datenregister von UART0 in Assembler ausgeben können – in `uartOut()` sollen Sie dies nun in C implementieren. Ein Pointer auf das Datenregister steht mit `UART0DR` schon zur Verfügung.

Vervollständigen Sie den Code überall so, dass Ihr `mpt5a.elf` schließlich den Hello-Text ausgibt.

Hinweis: Wenn Sie ein Programm einfach nur in seinem Ablauf ohne Debugger testen wollen, können Sie die Flags `-S` und `-s` beim `qemu`-Aufruf weglassen, dann läuft ihr Programm sofort und ungebremst ab. Wenn Sie mit dem Debugger arbeiten, können Sie natürlich `continue` und Breakpoints statt nur den Singlestep-Modus verwenden.

c) Das Gegenstück zu `uartOut()` bildet `unsigned char uartInWait()`, mit dem ein Zeichen eingegeben werden können soll. Lesen Sie die Anforderung im Quellcode-Kommentar und setzen Sie sie um.

Um auf die Ankunft eines Zeichens (ausnahmsweise in einer leeren Schleife) in `uartInWait()` zu warten, prüfen Sie wiederholt mit dem Flag `UART_RXFE` in Register `UART0FR`, ob das Empfangsregister noch leer ist. Falls nicht, können Sie das Zeichen aus `UART0DR` auslesen (kopieren) und als Funktionswert zurückgeben.

Testen Sie die Funktion, indem Sie jedes eingegebene Zeichen gleich wieder ausgeben (dieses Verhalten wird als *echo*-Modus eines Terminals bezeichnet).

d) Fügen Sie in `mpt5a.c` eine neue lokale Funktion `void uartPrintFlags(unsigned long flags)` hinzu. Diese Funktion bekommt den Inhalt des PSR (Program Status Register) übergeben und soll fünf Zeichen für die Flags N, Z, C, V, und Q ausgeben, die bei gesetztem Flag der Buchstabe des Flags, ansonsten ein Minus dastellen,

Beispiel: `-ZC--` oder `N--V-`.

Definieren Sie für die Implementierung auch Konstanten für die Stellen der Flags im PSR (als Bitmaske oder Bitnummer).

e) Mit den vorhandenen Funktionen sollten Sie nun in `mpt5a.c` folgendes Programmgerüst entwerfen und umsetzen:

- Es werden zwei achtstellige hexadezimale Zahlen (= jeweils 32 Bit) als Zeichenketten eingelesen (⇒ Array zum Speichern der Zeichen, verwendet in einer Schleife mit `uartInWait()` źum Einlesen).
- Während der Eingabe sollen die eingegebenen Zeichen auch angezeigt (echoed) werden (⇒ Ausgabe mit `uartOut()` in derselben Schleife).
- Die Zahlen sollen dann als Integer-Werte (nicht nur als Strings) in `unsigned long`-Variablen `a` und `b` konvertiert werden (⇒ `strtoul()` mit Basis 16).
- Schließlich sollen die beiden Zahlen binär ausgegeben werden (⇒ auch dafür gibt es eine vorbereitete Funktion).

- Sehen Sie auch erläuternde Zwischenausgaben vor, damit die Benutzer wissen, was sie tun sollen und was sie sehen.

f) Das eigentliche Ziel stellen anschließend die Funktionen `add()` und `adds()` dar. Sie werden von Ihnen in Assembler implementiert, aber dann später in `mpt5a.c` als C-Funktionen aufgerufen!

   Lesen Sie in `addfunc.h` und `addfunc.S`, was zu tun ist. Um herauszufinden, in welchen Registern die Argumente liegen, wenn Sie von C aus aufgerufen werden und in welches Register Sie den Rückgabewert legen müssen, orientieren Sie sich am "Procedure Call Standard for the ARM Architecture" (ARM AAPCS, s. LV-Server)

g) Erweitern Sie Ihr Programm in `mpt5a.c` so, dass die beiden eingegebenen Zahlen mit `add()` addiert werden und die dabei gesetzten Flags mit `adds()` bestimmt werden. Geben Sie das Ergebnis binär und die Flags mit `uartPrintFlags()` aus.

h) Überlegen Sie sich abschließend verschiedene Werte, deren Addition folgende Statusflags produziert: **N**egative, **Z**ero, O**V**erflow oder gar kein gesetztes Flag. Testen Sie durch Eingabe der Werte, ob Ihr Programm die Rechnung und Anzeige Ihren Erwartungen entsprechend durchführt.