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

Mikroprozessortechnik
SS 2020
LV 2522

Übungsblatt 7

Aufgabe 7.1 (Einfache Service Calls):

Sie nutzen in dieser Aufgabe den *Supervisor Call Handler* des Cortex-M3, um einfache Service Calls im Stil eines Betriebssystem-Dienstaufrufs umzusetzen. Es werden noch keine Privilegstufen genutzt, d.h. der Prozessor befindet sich nach wie vor stets im Privileged Mode.

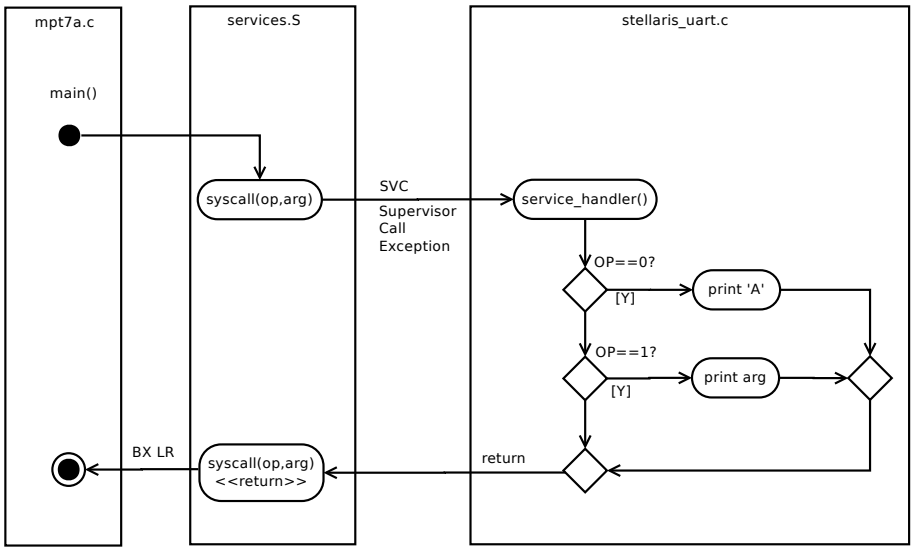
Auf C-Ebene soll zum Schluss eine Funktion `syscall(unsigned long op, unsigned long arg)` umgesetzt sein, bei der `op` angibt, welcher Service Call ausgeführt werden soll und mit `arg` ein weiteres Argument an den Service Call übergeben werden kann. Deshalb muss der Service Call Handler als Verteiler ("Dispatcher") zunächst `op` auswerten, um die gewünschte Operation anzuspringen.

Service Call 0 (also `syscall(0, ...)`) soll ein `A` auf der UART0-Schnittstelle ausgeben, Service Call 1 das in `arg` übergebene ASCII-Zeichen.

Auf Assemblerebene werden die Service Calls durch die ARM-Instruktion `SVC` umgesetzt. Die Instruktion verarbeitet eigentlich ein Argument für die Nummer des Service Calls, das allerdings nicht in einem Register oder auf dem Stack abgelegt wird, sondern umständlich aus dem Opcode von `SVC` extrahiert werden müsste.

Deshalb wird `op` vom Aufrufer einfach in `r0` abgelegt. `r1` enthält `arg`. Aus C heraus werden, durch `AAPCS` vorgegeben, diese beiden Register ohnehin für die ersten beiden Übergebaparameter eines C-Funktionsaufrufs verwendet, was wir uns als Vereinfachung zu Nutze machen. Um sich gegen zwischenzeitlich auftretende Interrupts abzusichern, müssten `r0` und `r1` eigentlich vom Stack geholt werden, wo sie der Cortex-M3 zu Beginn des Superviser Call Exception sichert; dieses Problem wird hier ignoriert.

Der Ablauf eines Aufrufs eines Service Calls nach diesem Design ist hier im Aktivitätsdiagramm (zur Vereinfachung statt einem eigentlich besser passenden Sequenzdiagramm) dargestellt.



a) Kopieren Sie Ihren Qellcode und das Makefile der letzten Aufgabe als Ausgangspunkt und benennen Sie `mpt7a.c`[1] in `mpt7b.c` um. Erzeugen Sie eine Assemblerdatei `services.S` als leere Datei und fügen Sie sie Ihrem Makefile hinzu.

b) Sei benötigen eine C-Funktion, die die Assembler-Instruktion `SVC` abbildet; sie soll daher selbst in Assembler in der Datei `services.S` implementiert sein und den Prototyp `void syscall(unsigned long svcno, unsigned long arg)` besitzen. Dies entspricht der Vorgehensweise für Blatt 4 mit `addfunc.S` und `addfunc.h`.

   Fügen Sie den Prototyp in `stellaris_uart.h` hinzu und eine Implementierung in services.S. Die Implementierung soll die Parameter ignorieren (diese bleiben in `r0` und `r1` "liegen" und werden dann im Supervisor Call Handler genutzt).

   Stattdessen ruft die Funktion direkt `SVC 0` auf (der Parameter `0` wird wie zuvor erwähnt nicht weiter verwendet, ist also eigentlich beliebig) und kehrt anschließend wie eine normale Subroutine zurück, ohne Schachtelung und Rückgabewerte zu berücksichtigen.

c) In `stellaris_uart.c` implementieren Sie den Handler mit dem Prototyp `void service_handler()`. Ergänzen Sie außerdem diesen Prototyp am Anfang von `startup.c` und fügen Sie den Funktionspointer als Eintrag an der korrekten Stelle in der Interrupt-Vektortabelle ein.

d) `service_handler()` ist in C implementiert und soll die in `r0` und `r1` übergebenen Argumente als lokale Variablen `op` und `arg` sehen können. Fügen Sie hierzu am Anfang der Funktion folgenden Quellcode ein, der dies ermöglicht:

```
unsigned int op,arg;

__asm("mov   %0, r0\n"
      "mov   %1, r1\n"
      :"=r" (op), "=r" (arg)::);
```

[1] In Aufgabenblatt 6 hatte sich ein Schreibfehler eingeschlichen: eigentlich hätte der Quellcode dort natürlich `mpt6a.c` heißen müssen.

Hinweis: Die Syntax und Nutzung der C-Variablenübergabe in `gcc`-Assembler wird an dieser Stelle vorgegeben, Sie sollten sich aber später in Ruhe die Details z.B. unter `https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html` ansehen.

Nach diesem Vorspann können sie die Inhalte von `r0` und `r1` als C-Funktionsparameter `op` und `arg` verwenden. Setzen Sie damit wie oben beschrieben die Operation 0 und die Operation 1 um.

Generell sollte ein Exception Handler schnell wieder zurückkehren; schreiben Sie daher den auszugebenden Wert direkt in `UART0DR`, statt `uartOut()` auzurufen.

e) Testen Sie den Service Call mit Testausgaben aus der `main()`-Funktion heraus.

f) Schreiben Sie schließlich die bisherigen `uart...()`-Ausgabefunktionen so um, dass sie den Service Call verwenden statt `uartOut()` aufzurufen, und erweitern Sie Ihre Testausgaben um Tests dieser modifizierten Funktionen.