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

Mikroprozessortechnik
SS 2020
LV 2522

Übungsblatt 6

Aufgabe 6.1 (Exception Handling):

In dieser Aufgabe werden die Mechanismen der Ausnahmebehandlung (Exception Handling) von Mikroprozessoren am Beispiel des ARM Cortex-M3 praktisch untersucht.

a) Ausgangspunkt ist Ihr Code von Aufgabenblatt 5. Legen Sie Kopien aller Quellcode-Dateien in einem neuen Verzeichnis an und benennen Sie `mpt5a.c` in `mpt7a.c` um. Passen Sie Ihr Makefile entsprechend an und testen Sie den bisherigen Stand noch einmal mit `qemu`.

b) Für den Anfang soll eine *Usage Fault*-Exception erzeugt werden. Ein einfacher Weg, diese Exception auszulösen ist der Versuch, einen illegalen Instruktions-Opcode, d.h. ein Wort im Speicher, das so als Instruktionswort nicht existiert, auszuführen. Wir setzen wir die illegale Instruktion in den Startup-Code noch vor dem Aufruf der `main()`-Funktion, da bis dahin noch keine nennenswerte Intialisierung des Prozessors erfolgt ist.

   Fügen Sie in den Code von `ResetHandler()` in `startup.c` vor den Aufruf von `main()` einen weiteren Assembler-Block ein (analog zum `asm__`-Block davor).

   Als einzige Instruktionszeile darin geben sie direkt einen konstanten Wert an, der im Speicher als (illegale) Instruktion gelesen werden wird:

```
".word 0xffffffff\n"  /* some illegal instruction  */
```

c) Schreiben Sie in `startup.c` eine neue C-Funktion `UsageFaultHandler()` als Exception Handler (analog `ISRDefaultHandler()` mit Endlosschleife) und tragen Sie den Pointer auf diese Funktion in der Verktortabelle für den Usage Fault Handler ein (die kommentierte Tabelle mit Funktionspointern ist ebenfalls in `startup.c` enthalten).

d) Bei der Ausführung in `qemu` unter Kontrolle von `gdb-multiarch` (oder `arm-none--eabi-gdb`) setzen Sie vorab einen Breakpoint auf `UsageFaultHandler()`. Führen Sie dann das Programm aus; dabei sollte der Breakpoint erreicht werden, sobald versucht wird, die illegale Instruktion auszuführen.

e) Untersuchen Sie nun am Breakpoint angelangt, wie Sie aus den Status- und Kontrollregistern des Cortex M3 die Ursache der Exception herausfinden könnten (wenn Sie sie nicht schon wüssten...).

Hinweis: Verwenden Sie das `gdb`-Kommando `x`, mit dem Sie den Inhalt von Speicherstellen ansehen können (Info via `help x`). Der Inhalt der Register `INTCTRL` und `FAULTSTAT` sollte Ihnen nützliche Informationen liefern. Informieren Sie sich um Datenblatt des TI Stellaris LM3S6965 über diese Register und Ihren Inhalt.

f) Um zu sehen, welche Register beim Einsprung in den Interrupt auf den Stack gerettet werden, erweitern Sie den Assembler-Block mit der illegalen Instruktion um *davor* stehende Instruktionen, mit denen Sie die Register `r0-r12` mit ausfsteigenden Werten von 1-13 laden, um sie zu "markieren" (0 kommt als Registerinhalt zu oft vor).

Führen Sie erneut bis zum Breakpoint aus und geben Sie die 9 zuoberst auf dem Stack liegenden Wörter aus.

Beantworten Sie: Welche Registerinhalte können Sie identifizieren? Vergleichen Sie mit den Informationen aus dem Abschnitt "Exception Entry" (2.5.7.1) im Stellaris-Datenblatt.

Warum müssen Sie 9 und nicht nur 8 Wörter betrachten (schauen Sie sich hierzu den Beginn des Exception-Handlers im Assemblercode an)?

Hinweis: Auch für die Ausgabe des Stack-Speichers bietet sich das `gdb`-Kommando `x` an. Nutzen Sie auch die Tatsache, dass Sie Registernamen mittels `$` als Argumente verwenden können, also z.B. `$sp` für den Stackpointer.

Aufgabe 6.2 (Test 2):

Die Ergebnisse zu den folgenden Testaufgaben müssen für die Berücksichtigung der Leistung bis zum Tagesende des 15.06.2020 per Email an `marcus.thoss@hs-rm.de` als PDF-Dokument mit dem Dateinamen `<Nachname>_<Vorname>_mpts20_t02.pdf` gesendet werden. Sofern nicht anders angegeben, wird für eine vollständig gelöste Teilaufgabe ein Punkt vergeben.

a) Beschreiben Sie, wie der Code beim Einsprung in eine Assembler-Subroutine für einen ARM Cortex M-3 aussehen muss, wenn die Routine am Ende mit

   1) `POP {R15}`
   2) `BX LR` (ohne vorheriges `POP {R15}`)

   verlassen wird und welchen Vorteil Variante 1) bietet.

b) In der Vorlesung haben Sie erfahren, dass die Aufrufkonventionen für die Register-Nutzung bei C-Funktionen für AVR-Prozessoren meist von der der `avr-libc` (der Implementierung der C-Standardbibliothek für gcc) übernommen werden. Lesen Sie die Konventionen im User Manual der `avr-libc` im Abschnitt *FAQ/What registers are used by the C compiler?* nach und beantworten Sie dann folgende Fragen (**2 P**):

   Betrachten Sie die C-Funktion
   `unsigned char f(unsigned char a, unsigned short b)`

1) In welchen Registern werden die Parameter a und b übergeben?
2) Welche Register dürfen nach dem Aufruf durch die Subroutine *nicht* von dieser verändert worden sein?

Hinweis:
Auch wenn das User Manual der `avr-libc` im Internet verfügbar ist — Sie finden es auch lokal in Ihrer Linux-Systeminstallation in verschiedenen Formaten im Verzeichnis `/usr/share/doc/avr-libc/`. Nutzen Sie im Zweifelsfall die Installation auf den Pool-PCs des Praktikums (z.B. `its01.local.cs.hs-rm.de`).

c) Beantworten Sie Frage b) für ARM-Prozessoren nach AAPCS-Konventionen (**2 P**).

d) Ordnen Sie die Fehlerbehandlungsstrategien a) Reset, b) Terminierung, c) Graceful Degradation und d) Recovery sinnvoll zu **(2P)**.

1) Beim Ausfall eines Binkers blinkt ein Auto stattdessen mit dem Tagfahrlicht.
2) Ein Prozess auf einem Linux-System greift auf eine unerlaubte Adresse zu.
3) Eine SMS kann momentan nicht gesendet werden.
4) Das Programm einer Waschmaschine ist noch vor dem ersten Wassereinlauf durch einen Spannungsimpuls aus dem Stromnetz gestört worden, so dass die Maschine blockiert.

e) Eine Sprunganweisung oder ein Software-Interrupt werden als als “synchron” angesehen. Synchron in Bezug worauf?

f) Durch welche Elemente wird das Sprungziel einer Interrupt-Anweisung bestimmt? Durch welche das einer Sprunganweisung im Assemblercode?

g) Welches sind die Interrupt-Ereignisse mit den drei höchsten (“wichtigsten”) Prioritäten beim ARM Cortex M-3? Was unterscheidet sie von allen anderen Interrupt-Vektoren?