SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Sebastian Hahn

Systemarchitektur SS 2016 – Projekt 2

Systemprogrammierung und Ausnahmebehandlung

Abgabemodalitäten

Das Projekt beginnt am 11. Juli 2016 mit Herausgabe dieser Beschreibung. Wir empfehlen das Projekt *so bald wie möglich* zu beginnen. Nutzen Sie die jeweiligen Officehours, sowie die Übungsgruppen um etwaige Verständnisprobleme und/oder Probleme mit dem MARS-Simulator auszuräumen.

Sie dürfen das Projekt entweder *selbstständig* oder in *Gruppen von zwei Personen* bearbeiten. Darüber hinaus ist Gruppenarbeit *nicht* erlaubt. Plagiierte Projekte werden wir mit **0 Punkten** bewerten. Wenn Sie eine Gruppe bilden möchten, laden Sie bitte in unserem CMS bis zum

13. Juli, 23:59 Uhr

eine Textdatei mit beiden Matrikelnummern (kommasepariert) hoch. Ansonsten werden Ihre Projekte getrennt behandelt.

Eine Person pro Gruppe muss die Ergebnisse Ihres Projekts bis zum

Montag, dem 25. Juli 2016, 23:59 Uhr

in unserem CMS-System hochladen. Insgesamt sind 20 Punkte (plus 16 Zusatzpunkte) zu erreichen. Zu spät abgegebene Projekte werden mit **0 Punkten** bewertet. Verwenden Sie zur Abgabe ein gzip-komprimiertes tar-Archiv, d.h. `*.tar.gz`. Das Archiv soll unmittelbar alle *bearbeiteten* Assemblerdateien enthalten. Verwenden Sie die von uns zur Verfügung gestellten Gerüst-Dateien, welche Sie in unserem CMS[1] finden. Falls Sie eine Aufgabe nicht bearbeiten, reichen Sie die zugehörige Assemblerdatei auch nicht ein.

Kommentieren Sie Ihren Code sinnvoll, da wir Ihre Abgaben manuell bewerten. D.h. beschreiben Sie innerhalb von Kommentaren die logischen Einzelschritte Ihrer Ausnahmebehandlung, nicht jedoch jede einzelne Instruktion. Orientieren Sie sich an den Kommentaren der Gerüstdateien. Unverständlicher und nicht kommentierter Code wird zu **Punktabzug** führen.

Werkzeuge und Dokumentation

Um Ihre Assemblerprogramme zu testen, verwenden Sie die von uns angepasste Variante des MARS-Simulator, die insbesondere Unterstützung für Timerinterrupts bietet. Diese Version **unterscheidet** sich sowohl von der Version unseres **ersten Projekts** als auch vom Original.

Die wichtigsten Informationen, die Sie für die Bearbeitung des Projekts benötigen, stellen wir im folgenden Abschnitt vor. Darüber hinaus bietet Teil 3 der offiziellen MIPS-Dokumentation zusätzliche Informationen[2] zur Ausnahme- und Unterbrechungsbehandlung. Unter [3] finden Sie, wie im ersten Projekt, eine Beschreibung aller MIPS-Instruktionen.

Hintergrundinformationen

Im folgenden finden Sie für das Projekt benötigte Hintergrundinformationen. In sämtlichen Aufgaben werden Sie jeweils eine Routine zur Ausnahmebehandlung schreiben. Solchen Routinen sind üblicherweise Bestandteil des Betriebssystems.

[1] `https://sysarch.cdl.uni-saarland.de/ss16/materials/index`
[2] `https://imgtec.com/?do-download=5145`
[3] `https://imgtec.com/?do-download=4287`

Eine Ausnahme (engl. *exception*) kann aus mehreren Gründen eintreten, z.B. durch ein Systemaufruf oder durch die Ausführung einer ungültigen Instruktion. Eine Ausnahme kann aber auch unabhängig vom ausgeführten Programm aufgrund von externen Geräten (Tastatur, Bildschirm) eintreten, z.B. wenn ein Zeichen auf der Tastatur eingegeben wurde. Solche Ausnahmen durch externe Geräte nennt man Unterbrechung (engl. *interrupt*).

Die Behandlung von Ausnahmen benötigt eine gewisse Hardwareunterstützung. In MIPS-Prozessoren stellt der sogenannte *System Coprozessor* diese Funktionalität zur Ausnahmebehandlung bereit. Ein Coprozessor hat allgemein eine eigene Menge von Registern und implementiert eigene zusätzliche Befehle. Er teilt sich aber auch Logik mit dem Prozessor, z.B. zum Fetchen von Instruktionen; daher Co-Prozessor.

Sobald eine Ausnahme eintritt, wechselt der Prozessor vom *user mode* zum sogenannten *kernel mode*. Im User Modus hat ein Programm nur eingeschränkten Zugriff auf die Maschine, z.B. kann es weder mit dem System Coprozessor kommunizieren noch bestimmte Instruktionen verwenden noch auf externe Geräte zugreifen. Im Kernel Modus hingegen hat ein Programm uneingeschränkten Zugriff auf die Maschine inklusive des System Coprozessors und externer Geräte.

Register des System Coprozessors

Die Register des Coprozessors 0 (CP0), des System Coprozessors, nennt man auch *special-purpose Register* im Gegensatz zu den 32 meist frei verwendbaren *general-purpose Register* des Hauptprozessors. Für unsere Zwecke der Ausnahmebehandlung betrachten wir die folgende Teilmenge der 32-Bit breiten special-purpose Register: *epc*, *status*, *cause* zur allgemeinen Ausnahmebehandlung und *count*, *compare* zur Programmierung von zeitgesteuerten, periodischen Interrupts (Timer interrupts).

Exception Program Counter *epc*

Kommt es zu einem Ausnahmefall, wird der aktuelle Programmzähler vom System Coprocessor in *epc* gesichert. Dieses Register enthält somit die Adresse der Instruktion, an welcher die Ausführung nach der Ausnahmebehandlung (meistens) fortgesetzt wird. Der *epc* ist CP0-Register 14 und kann gelesen und geschrieben werden.
Hinweis: Der MARS verwendet (standardmäßig) keine Delay Slots, was die Implementierung vereinfacht. Passagen in der MIPS-Dokumentation zu Delay Slots können Sie also ignorieren.

Status Register *status*

Das Status Register beinhaltet Informationen zum Zustand des System Coprozessors und ermöglicht eine gewisse Konfiguration der Ausnahmefunktionalität. Der *status* ist CP0-Register 12 und kann gelesen und geschrieben werden. Für uns sind lediglich die folgenden Bits relevant.

Bit 0: *Interrupt Enable IE* gibt an, ob Ausnahmen global erlaubt sind (1). Ist das Bit auf 0 gesetzt, werden keine Ausnahmen generiert.
Bit 1: *Exception Level EXL* gibt an, ob momentan eine Ausnahme behandelt wird. Während der Ausnahmebehandlung befindet sich der Prozessor im Kernel Mode.
Bit 4 - Bit 3: *KSU* gibt an in welchem Mode sich das System außerhalb der Ausnahmebehandlung befindet. Wir nehmen hier vereinfachend an, dass dies immer der *user mode* (Bits 10) ist.
Bit 15 - Bit 10: *Interrupt Mask IM[7] - IM[2]* gibt an ob auf Unterbrechungsanfragen von externen Geräten reagiert werden soll.

Cause Register *cause*

Das Cause Register beinhaltet zum Zeitpunkt einer Ausnahme den jeweiligen Grund für diese Ausnahme. Der *cause* ist CP0-Register 13 und kann *nur gelesen* werden. Für uns sind lediglich die folgenden Bits relevant.

Bit 6 - Bit 2: *ExcCode* gibt den Grund für die Ausnahme an. Eine Tabelle mit allen möglichen Belegungen finden Sie in Tabelle 9.53 auf Seite 212 der MIPS-Dokumentation Teil 3. Wichtig ist die Belegung 00000 – sie signalisiert einen Interrupt.
Bit 15 - Bit 10: *Interrupt Pending IP[7] - IP[2]* gibt an ob eine aktuelle Unterbrechungsanfrage eines externen Gerätes besteht. Die Positionen entsprechen den jeweiligen Positionen von *IM[7] - IM[2]* im Status Register.

Timer Interrupts

Unser MIPS-System unterstützt programmierbare, zeitgestützte Unterbrechungen. Der Timer wird über die Count und Compare Register gesteuert. Der Timer Interrupt ist assoziiert mit IP[7] und IM[7]. Das Bit IP[7] wird vom Timer auf 1 gesetzt sobald ein gewisser Zeitstempel erreicht ist, d.h. Compare und Count den gleichen Wert haben. Das Pending Bit IP[7] wird vom Timer wieder auf 0 gesetzt, sobald das Compare- oder Count-Register neu beschrieben wird.

Count Register *count* Das Count Register, CP0 Register 9, wird in jedem Zyklus um eins inkrementiert, es sei denn es wird mittels einer Instruktion beschrieben.

Compare Register *compare* Das Compare Register, CP0 Register 11, enthält einen Zeitstempel. Erreicht das Count Register diesen Zeitstempel, so wird ein Timer-Interrupt signalisiert. Das Register ist sowohl les- als auch schreibbar.

Kommunikation mit dem Coprozessor

Der Code der Ausnahmebehandlung selbst wird auf dem Hauptprozessor ausgeführt. Um innerhalb dieser Behandlungsroutine Zugriff auf die Register des Coprozessors zu haben gibt es zwei Instruktionen zur Kommunikation. Mithilfe von `mfc0` kann man den Wert eines special-purpose Registers in ein general-purpose Register kopieren und umgekehrt mit `mtc0` einen Wert in ein special-purpose Register kopieren.

Zusätzliche Befehle

Die Instruktion `eret` kehrt von der Routine zur Ausnahmebehandlung zum eigentlichen Programm zurück. Mithilfe der `syscall` Instruktion können Sie vom Nutzerprogramm aus einen Systemaufruf vom Betriebssystem anfordern. Es ist eine Konvention des sogenannten *Application Binary Interfaces* die Nummer des geforderten Systemaufruf in `$v0` und ein mögliches Argument in `$a0` zu übergeben. Für weitere Informationen, konsultieren Sie bitte die angegebene MIPS-Dokumentation.

Konvention: Reservierte General-Purpose Register

Obwohl alle 32 Register eines MIPS-Prozessors (mit Ausnahme des Register 0) frei verwendbar sind, gibt es bestimmte Konventionen bezüglich Ihrer Verwendung. Ein bereits bekanntes Beispiel ist Register 31 (genannt `$ra`) welches die Rücksprungadresse bei Funktionsaufrufen verwaltet. Eine für dieses Projekt wichtige Konvention ist es Register 26 (`$k0`) und 27 (`$k1`) für das Betriebssystem – z.B. unsere Ausnahmebehandlung – zu reservieren. Somit dürfen Register 26 und 27 *nicht* von gewöhnlichen Programmen verwendet werden. Einen kleinen Überblick liefert die Wikipedia-Seite[4].

Beispiel Ablauf Ausnahmebehandlung

Wir möchten folgendes Programm ausführen:

```
...
100: li $a0, 1234
104: li $v0, 1
108: syscall
10c: add $a0, $a0, 1
...
```

Wie sieht die Abarbeitung auf einer MIPS-Maschine aus? `//`-Kommentare beziehen sich auf die Hardware, `#` beziehen sich auf die Routine zur Ausnahmebehandlung.

[4] `https://en.wikipedia.org/wiki/MIPS_instruction_set#Compiler_register_usage`

```
...
108: syscall
// Generiere Ausnahme:
// - Setze epc auf 108
// - Setze kernel mode status[EXL] = 1
// - Setze cause[ExcCode] auf 8
// - Setze pc auf den Anfang der Routine zur Ausnahmebehandlung 0x80000180
# Führe Routine zur Ausnahmebehandlung aus
# - Sichere Register, die innerhalb dieser Routine verwendet werden
# - Untersuche ExcCode
# - Reagiere auf Exception/Interrupt (hier: gebe Zahl 1234 auf Konsole aus)
# - Addiere (ggfls.) 4 auf epc, hier: um Systemaufruf nicht zu wiederholen
# - Führe eret aus
// Kehre zu Programm zurück
// - Setze pc auf epc
// - Setze user mode status[EXL] = 0 auf
10c: add $a0, $a0, 1
...
```

Ausnahmegenerierung

Wir haben bereits den generellen Ablauf einer Ausnahmebehandlung gesehen, aber nicht wann eine Ausnahme generiert wird. Hierbei gibt es zwei Möglichkeiten. Erstens kann eine Ausnahme durch das ausführende Programm generiert werden, z.B. beabsichtigt durch einen Systemaufruf oder unbeabsichtigt durch eine Division durch 0. Zweitens kann eine Ausnahme durch die Unterbrechungsanfrage eines externen Gerätes generiert werden. Der Coprocessor generiert eine Ausnahme sobald eine nicht-maskierte Unterbrechungsanfrage vorliegt (*cause[15:10]* & *status[15:10]* $\neq$ 000000), interrupts global aktiviert sind (*status[IE]* = 1), und der Prozessor sich nicht innerhalb eines Ausnahmebehandlung befindet (*status[EXL]* = 0).

Memory-Mapped I/O Devices

Hier liefern wir etwas Hintergrundinformationen zu den externen Ein-/Ausgabegeräten. Ein bestimmter Teil Ihres Adressraums (von `0x00000000` bis `0xffffffff`) ist nicht mit Ihrem Hauptspeicher verbunden, sondern mit sogenannten *Ports* von externen Geräten. Sie können mit einem externen Gerät also durch Lesen und Schreiben auf die Adressen dieser Ports kommunizieren. Daher stammt der Name *memory-mapped input/output*.

Der MARS-Simulator stellt zwei (virtuelle) externe Geräte bereit: eine Tastatur und einen Bildschirm. Jedes dieser Geräte hat zwei Ports: einen Kontrollport und einen Datenport. Das unterste Bit des Kontrollports, genannt *ready*-Bit gibt an, ob das Gerät bereit zur Kommunikation ist. Das zweite Bit des Kontrollports, genannt *interrupt enable*-Bit, gibt an, ob das Gerät eine Unterbrechung generieren soll sobald das Gerät bereit wird. Das untere Byte des Tastatur-Datenports hält das letzte getippte Zeichen bereit. Das untere Byte des Bildschirm-Datenports kann mit dem nächsten auszugebenden Zeichen befüllt werden, wenn das Gerät bereit ist. Alle anderen Bits haben einen undefinierten Wert. Die Adressen der jeweiligen Ports finden Sie in Tabelle 1.

Das Verhalten der externen Geräten können Sie im MARS-Simulator unter `Tools->Keyboard and Display MMIO Simulator` beobachten (Bildschirm) und beeinflussen (Tastatur). Drücken Sie innerhalb des *Keyboard and Display Simulator* auf `Connect to MIPS` bevor Sie Ihre Programmausführung starten.

Sobald Sie eine (virtuelle) Taste drücken, wird das *ready*-Bit auf 1 gesetzt. Laden Sie das aktuelle Zeichen

Tabelle 1: Die Speicheradressen zur Kommunikation mit externen Geräten.

	Kontrollport	Datenport
Tastatur	`0xffff0000`	`0xffff0004`
Bildschirm	`0xffff0008`	`0xffff000c`

vom Datenport in ein Prozessorregister, wird das *ready*-Bit zurück auf 0 gesetzt. Sobald Sie ein Zeichen in den Datenport des (virtuellen) Bildschirms laden, wird das *ready*-Bit auf 0 gesetzt. Nach einer (konfigurierbaren) Verzögerung erscheint das Zeichen auf dem Bildschirm und das *ready*-Bit wird wieder auf 1 gesetzt. Ist ein externes Gerät *ready* und das jeweilige *interrupt enable*-Bit gesetzt, wird eine Unterbrechungsanfrage gestellt. Hierzu wird das jeweilige interrupt-pending Bit im *cause*-Register auf 1 gesetzt, *IP[2]* für die Tastatur und *IP[3]* für den Bildschirm. Das interrupt-pending Bit wird 0 sobald das entsprechende externe Gerät nicht mehr *ready* ist oder das *interrupt enable*-Bit auf 0 gesetzt wird.

Aufgabe 2.1: Systemaufrufe — 4 Punkte

Implementieren Sie den Systemaufruf mit Nummer 11, welcher das in `$a0` übergebene ASCII-Zeichen auf dem externen Bildschirm ausgibt. Sie können analog zum Abschnitt *Beispiel Ablauf Ausnahmebehandlung* vorgehen. Für andere Ausnahmen sowie andere Systemaufrufe soll Ihre Ausnahmebehandlung nichts tun und einfach zum Userprogramm zurückkehren.

*Hinweis: Zur Ausgabe dürfen Sie eine Warteschleife (*busy wait*) verwenden, welche darauf wartet, dass das* ready*-Bit des Bildschirms gesetzt wird. Insbesondere müssen Sie keine Interrupt-Behandlung für den Bildschirm programmieren.*

Aufgabe 2.2: Prozesswechsel — 16 Punkte

In dieser Aufgabe soll ein periodischer Prozesswechsel zwischen zwei nicht-kooperativen Programmen implementiert werden. Die beiden vorgegebenen Programme sollen abwechselnd für jeweils circa 100 Takte ausgeführt werden.

Überlegen Sie sich zunächst wie Sie periodisch die Kontrolle des Prozessors an das Betriebssystem übergeben können. Im zweiten Schritt überlegen Sie sich, wie Sie den eigentlichen Prozesswechsel ausführen können. Betrachten Sie hierzu die jeweiligen Prozesskontrollblöcke. Sie müssen den vorgegebenen Kontrollblöcken noch Felder hinzufügen. Löschen Sie keine der vorgegebenen Felder. Fügen Sie schließlich beide Teile zusammen und implementieren Sie eine entsprechende Ausnahmebehandlung in MIPS-Assembler.

Stellen Sie durch Simulation mit dem MARS-Simulator sicher, dass beide Programme abwechselnd ausgeführt werden. Insbesondere sollen Sie den Code der beiden vorgegebenen Prozesse *nicht* ändern.

Hinweis: Es ist ausreichend die Register im Kontrollblock zu sichern, die tatsächlich von unseren User-Programmen verwendet werden.

Aufgabe 2.3: Bonus: Memory-Mapped Ein-/Ausgabe — 4+12=16 Bonuspunkte

In dieser Aufgabe soll die Kommunikation mit externen Ein-/Ausgabegeräten realisiert werden, genauer mit einer Tastatur und einem Bildschirm. Schreiben Sie ein Programm, welches Zeichen von der Tastatur des *Keyboard and Display Simulator* einliest und in der eingegebenen Reihenfolge auf dem dortigen Bildschirm wieder ausgibt. Wenn die Tastatur Eingaben schneller liefert als der Bildschirm Ausgaben verarbeiten kann, dürfen Sie zusätzliche Eingaben verwerfen.

1. Implementieren Sie die Funktionalität zunächst mit *Polling*. Beim *Polling* fragen Sie in einer Schleife ständig den Status Ihrer Geräte ab (engl. *to poll*) und handeln dementsprechend. Insbesondere kommen keine Interrupts zum Einsatz.

2. Implementieren Sie die oben beschriebene Ein-/Ausgabe-Funktionalität mithilfe von *Interrupts*. Im Gegensatz zum *Polling*, lassen Sie sich hier von Statusänderungen Ihrer externen Geräte via Interrupts benachrichtigen. Es wird also nur im Falle einer Statusänderung Arbeit verrichtet. Sie werden einen Puffer zwischen Ein- und Ausgabe benötigen. Verwenden Sie einen 16 Bytes großen Puffer. Ein Byte davon darf immer unbenutzt bleiben.

Hinweis: Wählen Sie im `Keyboard and Display Simulator` *einen hohen Wert für die Transmitter Delay Length um die tatsächliche Langsamkeit von I/O-Geräten nachzuempfinden.*