RA FS 12 Serie 4

Samuel Bucheli, Gian Calgeer, Judith Fuog

Die vierte Serie ist bis Donnerstag, den 3. Mai 2012 um 17:00 Uhr zu lösen und in schriftlicher Form in der Übungsstunde abzugeben. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung. Allfällige unlösbare Probleme sind uns so früh wie möglich mitzuteilen, wir werden gerne helfen.
Viel Spass!

Theorieteil

Gesamtpunktzahl: 12 Punkte

1 Sign Extension (1 Punkt)

Erläutern Sie, weshalb die Sign-Extension gerade bei Branch-, Load- und Store-Befehlen gebraucht wird.

2 Logische und bitweise Operationen (1 Punkt)

Bestimmen Sie die jeweilige Ausgabe:

Eingabe	Ausgabe
`0xFE & 0xEF`	
`0xFE && 0xEF`	
`0xFE | 0xEF`	
`0xFE || 0xEF`	
`~0xFE`	
`!0xFE`	

3 Hardwarezugriff (1 Punkt)

Wie könnte ein Prozessor auf verbundene Hardware zugreifen?
Benutzen Sie die MIPS Emulation als Inspiration.

4 `bne` statt `beq` (2 Punkte)

Welche Änderungen sind bei der in der Vorlesung vorgestellten MIPS-Implementation (siehe "Basic MIPS Architecture Review") notwendig, um `bne` statt `beq` zu implementieren.

(a) Für die Singlecycle Implementation

(b) Für die Multicycle Implementation

5 Pipeline Registers (1 Punkt)

Wozu werden die Register (siehe Folie 6, "Basic MIPS Pipelining Review") zwischen den einzelnen Berechnungsstufen benötigt?

6 Pipelining Hazard (2 Punkte)

Erklären Sie den Unterschied zwischen Control-, Data- und Structural-Hazards.

7 Stall (2 Punkte)

Erläutern Sie, warum es auf Folie 15, im Gegensatz zur Folie 19, genügt, nur zwei Taktzyklen zu warten. (Die Seitenangaben beziehen sich auf das Kapitel "Basic MIPS Pipelining Review")

8 Data Hazard (2 Punkte)

Geben Sie sämtliche Data Hazards im folgenden Code an. Bei welchen Abhängigkeiten handelt es sich um Data Hazards, die mittels Forwarding behoben werden können? Bei welchen Abhängigkeiten handelt es sich um Data Hazards, die zu einem *stall* führen?

```
1 add $t1, $t2, $t3
2 sub $t4, $t1, $t2
3 lw $s2, 200($t1)
4 add $s3, $t1 ,$s2
```

Optionale Fragen

Die folgenden Fragen beziehen sich auf die Dateien aus dem Programmierteil, sie müssen nicht beantwortet werden, helfen aber beim Verständnis des Programmierteils.

`exception.s`

Wozu dient der Code in `exception.s`?

Interval Timer

Wie wird die aktuelle "Zeit" festgehalten und jeweils aktualisiert?

`movia`

Was bewirkt der Befehl `movia`? Wie und wo ist dieser implementiert?

Programmierteil

(a) Stellen Sie zuerst sicher, dass Ihr Altera Board korrekt funktioniert und dass Sie in der Lage sind, Assembler-Code zu kompilieren und auszuführen. Eine entsprechende Anleitung befindet sich auf Ilias.

Stellen Sie einen Defekt beim Altera Board fest, melden Sie sich bitte umgehend bei den Assistenten.

(b) Machen Sie sich mit dem "DE1 Basic Computer" und dem "Altera Nios II Soft Processor" vertraut, lesen Sie dazu die Dokumente `DE1_Basic_Computer` und `tut_nios2_introduction`. Diese befinden sich auf Ilias.

(c) Studieren Sie das Programm zum Systemtest, studieren Sie ebenfalls die Programme, die in in der Dokumentation vorgestellt werden.

Hinweis: Es empfiehlt sich ebenfalls, diese mit dem *Altera Monitor Program* schrittweise auszuführen

(d) Laden Sie das zur Übungsserie gehörende Programmgerüst von Ilias herunter, studieren Sie dieses[1]. Die optionalen Fragen können beim Verständnis des Programmgerüsts helfen.

Ihre Aufgabe ist es, das gegebene Programmgerüst wie folgt zu vervollständigen:

(a) Laden Sie die zur Serie 4 gehörigen Dateien von Ilias herunter und studieren diese aufmerksam.[2] Versuchen Sie zu verstehen, was die bereits vorhandenen Teile bedeuten, die optionalen Fragen können Ihnen dabei helfen.

(b) Tragen Sie Ihren Namen sowie den Namen einer allfälligen Übungspartnerin oder eines allfälligen Übungspartners an den vorgesehenen Stelle in den Dateien`knightRider.s` und `pushbutton.s` ein.

(c) Implementieren Sie auf `LEDR0..LEDR9` ein Lauflicht, das bei `LEDR0` beginnt, dann nach `LEDR9` wandert, dann wieder zurück nach `LEDR0` und so weiter (in `knightRider.s`).

(d) Erweitern Sie Ihr Programm, so dass das Drücken von `KEY3` eine Beschleunigung und `KEY1` eine Verlangsamung der Laufgeschwindigkeit des Lichtes bewirkt (in `pushbutton.s`).

Das Drücken von `KEY2` soll die Geschwindigkeit auf den ursprünglichen Wert zurücksetzen.

(e) Stellen Sie sicher, dass das Assemblerprogramm *ausführlich und sinnvoll* kommentiert ist. Als Richtwert gilt, dass jede Zeile kommentiert werden soll, man kann zwei Zeilen zusammenfassen, falls dies Sinn macht.

Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.

(f) Stellen Sie sicher, dass Ihre Implementation ohne Fehler und Warnungen kompilierbar ist.

Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.

[1] `configuration.ncf` und `nios_system.ptf` ausgenommen
[2] `configuration.ncf` und `nios_system.ptf` ausgenommen

(g) Erstellen Sie aus Ihrer Lösung eine Zip-Datei namens `<nachname>.zip` (wobei `<nachname>` natürlich durch Ihren Nachnamen zu ersetzen ist).

(h) Geben Sie die Datei elektronisch durch Hochladen in Ilias ab.

(i) Drucken Sie *zusätzlich* die die Dateien `knightRider.s` und `pushbutton.s` aus und geben Sie diese in der Übungsstunde zusammen mit der restlichen Serie 4 ab.