

Prof. L. Thiele

## Technische Informatik 1 - HS 2011

---

### Übung 7

Datum: 24.11.2011

---

## 1 Pipelining

In dieser Übung soll mit Hilfe des Simulators WinMIPS64 die Programmausführung in einer RISC-Pipeline untersucht werden. WinMIPS64 simuliert die 64-Bit Erweiterung der in der Vorlesung behandelten MIPS-Architektur.

Für diese Übung wird der 64-Bit MIPS Instruktionssatz verwendet. Im Vergleich zu dem aus der Vorlesung bekannten 32-Bit Instruktionssatz sind folgende Unterschiede relevant.

- Die 32 Register ( $r_0 - r_{31}$ ) sind 64 Bit breit.  $r_0$  ist fest mit 0 verdrahtet ( $r_0 \equiv 0$ ).
- DADDI,  $\$t, \$s, imm$ : Doubleword Add Immediate (64-Bit Variante von ADDI).
- LD  $\$t, offset(\$s)$ : Load Doubleword (64-Bit Variante von LW).
- SD  $\$t, offset(\$s)$ : Store Doubleword (64-Bit Variante von SW).

Das Tool WinMIPS64 ist unter folgender Adresse verfügbar:

[http://www.tik.ee.ethz.ch/tik/education/lectures/II1/materials/exercise\\_07.zip](http://www.tik.ee.ethz.ch/tik/education/lectures/II1/materials/exercise_07.zip)

WinMIPS64 ist eine Windows Anwendung. Falls Sie ein Windows Notebook dabei haben, empfiehlt es sich WinMIPS64 darauf auszuführen. Starten Sie dazu `winmips64.exe` aus dem oben genannten zip Archiv (es ist keine Installation notwendig).

Geben sie folgende Befehle ein, um WinMIPS64 unter den Tardis Maschinen auf Linux auszuführen (dazu wird die `wine` Laufzeitumgebung verwendet):

1. Entpacken des Archivs ins Home Verzeichnis: `unzip exercise_07.zip -d ~`
2. Wechseln in das neu erstellte Verzeichnis: `cd ~/exercise_07`
3. Setzen von Ausführungsrechten für das Setup-Skript: `chmod 755 setup.sh`
4. WinMIPS64 installieren und ausführen: `./setup.sh`  
(WinMIPS64 wird dabei ins Verzeichnis `~/wine/drive_c/winmips64` installiert.)
5. WinMIPS64 startet automatisch. Wenn nicht, folgenden Befehl ausführen:  
`wine c:\\winmips64\\winmips64.exe`

Die Anwendung des WinMIPS64 Simulators ist dank der graphischen Benutzeroberfläche intuitiv und bedarf keiner besonderen Vorkenntnisse. Bei Unklarheiten empfiehlt sich folgendes Tutorial zu lesen:

[http://www.tik.ee.ethz.ch/tik/education/lectures/II1/materials/...  
exercise\\_07\\_winmips64tut.pdf](http://www.tik.ee.ethz.ch/tik/education/lectures/II1/materials/...exercise_07_winmips64tut.pdf)

Gegeben sei folgendes Programm für den MIPS64-Prozessor, welches die 4 Elemente des Arrays DATA um 1 inkrementiert. Die Elemente des Arrays sind 64 Bit breit.

```

.data
DATA: .word 12,13,14,15

.code
main:
    daddi    r17,r0,DATA    #save address of array DATA to register r17
    daddi    r8,r0,4
loop:
    ld      r9,0(r17)
    daddi   r9,r9,1
    sd      r9,0(r17)
    daddi   r17,r17,8
    daddi   r8,r8,-1
    bnez   r8,loop
    halt                    #stop execution

```

### 1.1 Kurze Verständnisfragen

- (a) Welche Funktion hat das Register r8? Welchen Wert muss es vor dem Schleifendurchlauf erhalten, falls das Array 30 Elemente enthält?
- (b) Warum wird beim Schreiben der Arraywerte in den Speicher der Offset in 8er-Schritten erhöht?
- (c) Übersetzen Sie das obige Programm sinngemäss nach C.

### 1.2 Pipelining mit und ohne Forwarding

WinMIPS64 stellt Stalls abhängig von ihrer Ursache in zwei Varianten dar (siehe Tabelle 1): (1) muss auf Daten von einer vorhergehenden Instruktion gewartet werden (Daten Hazard), wird die Stufe solange wiederholt bis die korrekten Werte am Eingang der Stufe vorliegen. Die Wiederholung der Stufe wird dabei mit raw gekennzeichnet. (2) Muss eine Instruktion durch einen Stall einer vorhergehenden Instruktion blockiert werden, wird die Wiederholung der Stufe mit einem leeren Feld gekennzeichnet.

daddi r1,r0,1	IF	ID	EX	MEM	WB				
sd r1,0(r2)		IF	ID	raw	raw	EX	MEM	WB	
daddi r3,r3,4			IF			ID	EX	MEM	WB

Tabelle 1: Darstellung von Stalls in WinMIPS64.

In der Vorlesung sowie der Musterlösung wird folgende vereinfachte Darstellung verwendet (Tabelle 2). Dabei wird die Pipeline Stufe eines Befehls erst dann ausgeführt wenn das korrekte Eingangssignal vorliegt.

daddi r1,r0,1	IF	ID	EX	MEM	WB				
sd r1,0(r2)		IF	-	-	ID	EX	MEM	WB	
daddi r3,r3,4					IF	ID	EX	MEM	WB

Tabelle 2: Vereinfachte Darstellung von Stalls.

Gehen Sie folgendermassen vor, um das Programm in WinMIPS64 zu laden:

1. Laden des Programmes: Im WinMIPS64 Menü *File* → *Open*. Öffnen sie die Datei `~/ .wine/drive_c/winmips64/ex7_mips64.s`.
  2. Ausführen des Programmes: Benutzen Sie dazu das WinMIPS64 Menü *Execute*.
- (a) Simulieren Sie das Pipelineverhalten **ohne Data Forwarding** für die ersten zwei Iterationen der Schleife. *Enable forwarding*, *Enable Branch Target Buffer* sowie *Enable Delay Slot* unter *Configuration* müssen ausgeschaltet sein.
- I. Geben Sie das Pipelining-Diagramm in der vorbereiteten Tabelle 3 an.
  - II. Welchen Anweisungen führen zu Wartezyklen? Erklären Sie deren Zustandekommen. Geben sie ausserdem an, ob es sich um Strukturelle, Daten oder Ablauf Hazards handelt.
  - III. Wie viele Taktzyklen werden für die Ausführung des kompletten Programms insgesamt benötigt? Wie viele Instruktionen werden dabei abgearbeitet? Wie gross ist der CPI (cycles per instruction)?
  - IV. In der WB Stufe werden die Resultate ins Register zurück geschrieben, während die ID Stufe die für die Instruktion benötigten Werte aus dem Register liest. Wieso kann bei der Ausführung der `addi r9,r9,1` Instruktion die ID Stufe gleichzeitig zur WB Stufe der vorhergehenden Instruktion `ld r9,0(r17)` geschehen (beide Instruktionen greifen auf das Register `r9` zu)?
- (b) Simulieren Sie das Pipelineverhalten **mit Data Forwarding** für die ersten zwei Iterationen der Schleife. *Enable Branch Target Buffer* sowie *Enable Delay Slot* sollen weiterhin ausgeschaltet sein. Hinweis: In WinMIPS64 sind die Register welche für das Forwarding potentiell zur Verfügung stehen farbig markiert. Die Farbe entspricht der Pipeline Stufe (des vorhergehenden Zyklus) welche den Wert des Registers zur Verfügung stellt.
- I. Geben Sie das Pipelining-Diagramm in der vorbereiteten Tabelle 4 an. Zeichnen Sie mittels Pfeilen im Diagramm ein, wo der Data-Forwarding-Mechanismus eingesetzt wird.
  - II. Wie viele Taktzyklen werden für die Ausführung des Programms insgesamt benötigt? Wie viele Instruktionen werden dabei abgearbeitet? Wie gross ist der CPI (cycles per instruction)?
  - III. Wieso gibt es bei der Instruktion `bnez r8,loop` trotz forwarding einen Stall?

### 1.3 Pipelining mit Forwarding und Delay Slot

Die Pipeline realisiere zusätzlich einen Branch Delay mit einem Delay Slot und implementiere Data Forwarding wie im Skript beschrieben.

- (a) Optimieren Sie die angegebene Programmschleife mittels Umordnen der Instruktionen so, dass unter Ausnutzung des Delay Slots die Ausführungszeit weiter reduziert werden kann.
- (b) Benutzen Sie WinMIPS64 um die Lösung zu verifizieren. Die Funktion *Enable Delay Slot* muss dabei aktiviert werden. Insbesondere soll darauf geachtet werden, dass am Ende des Programmes das korrekte Resultat im Array steht (siehe Programmfenster Data welches die Werte im Speicher hexidezimal darstellt). Wie viele Zyklen dauert die Ausführung des Programmes? Wieviele Stalls treten dabei auf?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
daddi r17,r0,DATA																
daddi r8,r0,4																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																

4

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
daddi r17,r0,DATA																
daddi r8,r0,4																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																

Tabelle 3: Pipelining Diagramm für Ausführung ohne Data Forwarding

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
daddi r17,r0,DATA																
daddi r8,r0,4																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																

5

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
daddi r17,r0,DATA																
daddi r8,r0,4																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																
ld r9,0(r17)																
daddi r9,r9,1																
sd r9,0(r17)																
daddi r17,r17,8																
daddi r8,r8,-1																
bnez r8,loop																
halt																

Tabelle 4: Pipelining Diagramm für Ausführung mit Data Forwarding