



**Übungen zur Vorlesung
“Database Systems and Modern CPU Architecture”
Sommersemester 2009**

Jan Rittinger (jan.rittinger@uni-tuebingen.de)

3. Übungsblatt

Ausgabe: 13. Mai 2009 · Besprechung: 20. Mai 2009

Bitte schicken Sie die Lösung der Aufgaben, die mit „**(Abgabe)**“ markiert sind (bis spätestens 15 Minuten vor der Übung) an jan.rittinger@uni-tuebingen.de. Die Abgabe dient als Einschätzung für die „aktive“ Teilnahme an den Übungen und als bessere Diskussionsgrundlage für die Übungsbesprechung.

Aufgabe 1: SPIM—Ein MIPS-Simulator

Ein einfacher Simulator eines MIPS-Prozessors ist mit SPIM unter <http://www.cs.wisc.edu/~larus/spim.html> frei verfügbar. Ähnlich eines Debuggers lassen sich einzelne MIPS-Assembleroperationen Schritt für Schritt ausführen und deren Effekte auf die CPU-Register beobachten.

Installieren Sie SPIM auf Ihrem System. Unter der obigen lassen sich Binaries und Quellcode für verschiedene Plattformen herunterladen.

Hinweise:

- SPIM kann direkt Textdateien einlesen, die mnemonischen Assemblercode beinhalten. Im Unterschied zu anderen Emulatoren brauchen Sie den Assemblercode nicht zuerst (mit einem Crosscompiler) zu kompilieren.
- Umfangreiche Dokumentation zu SPIM und seinem Assembler-Dialekt finden Sie im Internet (http://pages.cs.wisc.edu/~larus/SPIM/spim_documentation.pdf). Achten Sie darauf, dass die Syntax von SPIM sich in einigen Details geringfügig von der auf den Vorlesungsfolien gezeigten unterscheidet.

Aufgabe 2: MIPS-Code: Fakultät

(Abgabe)

Schreiben Sie ein SPIM Programm das die Fakultät eines im Memory abgelegten Wertes ermittelt und das Ergebnis ausgibt.

Aufgabe 3: MIPS: ???

Gegeben sei der folgende Quellcode für SPIM:

```
1      .text
2      .globl __start
3
4  __start:
5      lw      $a0, input
6
7      move    $v0, $a0
8      blt    $a0, 2, bar
9      li     $t0, 0
10     li     $v0, 1
11  foo:    add    $t1, $t0, $v0
12         move  $t0, $v0
13         move  $v0, $t1
14         sub   $a0, $a0, 1
15         bgt  $a0, 1, foo
16  bar:    sw     $v0, result
17
18         li   $v0, 10          # exit program
19         syscall
20
21     .data
22  input: .word  0x00000008
23  result: .word 0x11111111
```

1. Was berechnet dieses Programm?
2. Welche Unterschiede zwischen dem Assemblercode und den ausgeführten Instruktionen lassen sich in diesem Programm beim Laden in SPIM beobachten?