



Technische Informatik II im SS 2007

Aufgaben zu den Tutorien in der Woche
vom 29. bis 31. Mai 2007

Prof. Dr. J. Henkel
Dr.-Ing. Tamim Asfour

Haid-und-Neu-Str. 7
2. OG., Raum 313.1
D-76131 Karlsruhe

Telefon: +49-721-608-7379
Fax: +49-721-608-8270
Email: asfour@ira.uka.de
<http://i61www.ira.uka.de/users/asfour/TI>

1 Was ist SPIM?

SPIM ist ein Simulator eines virtuellen Rechners, auf dem man Programme für den MIPS-R2000-Prozessor ablaufen lassen kann. Der Simulator bzw. der MIPS-Assembler nimmt die 1:1-Übersetzung der symbolischen Befehle in Maschinencode vor. Zusätzlich erlaubt er komplizierte Details vor dem Programmierer zu verbergen, die bei der Programmierung eines RISC-Rechners zu berücksichtigen sind und welche die Programmierung eines MIPS-Rechners schwierig machen. Dazu gehören:

- *delayed instructions*: Verzögerte Ausführung von Lade- und Sprungbefehlen, da diese zwei Taktzyklen zu ihrer Ausführung benötigen (die prozessorinterne Pipeline läuft weiter). Im zweiten Taktzyklus wird bereits die nächste Instruktion dekodiert und abgearbeitet. Dies hat zur Folge, daß eine Instruktion, die hinter einem Lade- oder Sprungbefehl steht, vor dem eigentlichen Sprung bzw. Erscheinen des geladenen Wertes ausgeführt wird. Der Assembler verbirgt dies durch die Vertauschung der Befehlsreihenfolge bei verzögerten Instruktionen.
- **Eingeschränkter Befehlssatz**: Der Assembler erweitert den Befehlssatz durch Bereitstellung von Pseudo-Instruktionen, die auf eine oder mehrere echte Maschinen-Instruktionen abgebildet werden.
Beispiel: Der Vergleich auf > (**sgt**) durch einen Vergleich auf < (**slt**) durch Vertauschung der Operanden.
- **Beschränkungen des Befehlsformats**: Bei Instruktionen mit einem direkten Operanden (**imm**) darf dieser eine maximale Größe von 16 Bits haben. Der Assembler fügt bei größeren Werten zusätzliche Ladebefehle ein.

2 Assemblerdirektiven oder Assembleranweisungen

Assemblerdirektiven oder Assembleranweisungen dienen zur Angabe von Zusatzinformationen, wie Speicherplatzreservierung für unterschiedliche Datentypen (ab der aktuellen Adresse), Segmentzuordnung, ... usw.

```
.align n, .data, .text, .globl, .kdata, .ktext,  
.ascii, .asciiz, .byte, .half, .word, .float, .double, .space
```

Daten und Programmcode *müssen* in separaten Speicherbereichen untergebracht werden. Datendefinition und Speicherreservierung müssen im Datensegment stehen. Befehle im Textsegment. Die Zuordnung in ein Segment erfolgt implizit über die gerade aktive Segmentangabe, die mit Hilfe der Anweisungen `.data` auf das Datensegment und `.text` auf das Textsegment gesetzt werden kann.

3 Systemaufrufe (*system calls*)

Der Aufruf von Systemfunktionen erfolgt über die Instruktion `syscall`. Jeder Systemaufruf wird über eine eindeutige Nummer (*call code*) identifiziert, die man vor dem Aufruf ins Register `$v0` laden muss. Operanden werden in den Register `$a0` bis `$a3` erwartet. Fließkommawerte im Register `$f12`. Die Ergebnisse eines Aufrufs werden im Register `$v0` bzw. `$f0` zurückgegeben.

4 Beispielprogramm

```
# Hello World im MIPS-R2000-Assembler

        .data                                # Es folgen Daten im Datensegment
hello:   .asciiz "Hello World! \n"          # auszugebender Text

        .text                                # Umschalten aufs Textsegment
        .globl main                         # main als globales Symbol

main:    li $v0, 4                           # Systemaufruf-Nr. für print_string -> $v0
        la $a0, hello                       # Adresse des auszugebenen Texts -> $a0
        syscall                             # Systemaufruf
        jr $ra                              # Programmende
```

SPIM führt beim Starten eines geladenes Programm einen Sprung zur Adresse mit dem symbolischen Namen `main` aus. Deshalb muß jedes Programm in seinem Textsegment (Codesegment) die Sprungmarke `main` definieren.

Im Tutorium soll der Umgang mit den Assemblerdirektiven und Systemaufrufen behandelt werden. Entsprechende Beispiele zu den anderen Systemaufrufen analog zum obigen Beispiel.

Aufgabe 1

- Schreiben Sie die folgenden Kontrollstrukturen in MIPS-Assembler um. Sie dürfen nur die MIPS-Befehle `slt`, `beq` und `bne` verwenden. Zur Speicherung temporärer Variablen verwenden Sie das Register `$at`. Die Variable `a` ist im Register `$t4`, die Variable `b` im Register `$s0`.

```
(a)    if ( a <= b )
        {
            ...
        }
marke1:
```

```
(b)   if ( a >= b )
        {
            ...
        }
    marke2:
```

```
(c)   do
        {
            marke3:
                ...
        } while ( a != b )
```

2. Was sind die Unterschiede zwischen einer statischen Speicherallokierung und einer dynamischen Speicherallokierung?