

## Rechnerarchitektur im Sommersemester 2018

### Übungsblatt 6

**Abgabetermin:** 28.05.2018, 12:00 Uhr

**Besprechung:** Besprechung der T-Aufgaben in den Tutorien vom 22. – 25. Mai 2018  
Besprechung der H-Aufgaben in den Tutorien vom 28. Mai – 01. Juni 2018

#### Aufgabe 27: (T) Umsetzung Boolescher Ausdrücke

(– Pkt.)

Übersetzen Sie folgendes Pseudocodefragment in MIPS-Code. Gehen Sie davon aus, dass der Wert der Variablen *a* bereits in das Register *\$t0* geladen wurde.

```
1 IF (a < 0) OR (a > 99) THEN
2   a := a - 10;
3 ELSE
4   a := a - 1;
5 END;
```

Bedenken Sie dabei insbesondere: Der Ausdruck *a > 99* wird nur dann ausgewertet, wenn *a < 0* fehlgeschlagen ist.

#### Aufgabe 28: (T) SPIM Programmieraufgabe

(– Pkt.)

Erstellen Sie ein **vollständiges** SPIM-Programm, das folgendes durchführt:

- Es werden zwei positive Integer-Zahlen von der Konsole eingelesen.
- Es wird der Durchschnitt dieser beiden Zahlen auf eine Nachkommastelle genau berechnet.
- Das Ergebnis der Berechnung wird ausgegeben.

**Tipp:** Programmieren Sie diejenigen Schritte, die Sie auch beim handschriftlichen Dividieren durchführen!

**Beachten Sie hierbei folgendes:**

- Verwenden Sie nur die **unten aufgeführten Befehle**.
- Verwenden Sie für die Vorkommazahl das Register *\$s0* und für die Nachkommazahl das Register *\$s1*, ansonsten nur die temporären Register.
- **Kommentieren** Sie ihr Programm sinnvoll!
- Sowohl die Eingabe als auch die Ausgabe soll mit einem Anweisungstext versehen werden, wie z.B. *”Geben Sie die 1. Zahl ein: ”*, etc.

## Aufgabe 29: (H) Test des MIPS Simulators

(8 Pkt.)

Für diese Aufgabe sollten Sie sich mit dem MIPS-Simulator SPIM vertraut machen. Sie können einen MIPS-Simulator von der Vorlesungshomepage herunterladen, oder XSPIM aus dem CIP-Pool benutzen.

- Laden Sie sich das Assemblerprogramm `simple.s` von der Rechnerarchitektur-Homepage herunter und speichern Sie es in Ihrem Home-Verzeichnis ab.
- Starten Sie Ihren Simulator. Im CIP-Pool: `xspim` auf der Konsole eingeben.
- Laden Sie das Programm `simple.s` in den Simulator und führen Sie es aus. Dabei sollte eine Konsole erscheinen, über die die Ein- und Ausgabe erfolgt.

Beantworten Sie nun folgende Fragen.

- a. Welches Ergebnis liefert das Programm für die Eingabefolge "1, 2, 3, 4, 0"? (D.h. nach Start des Programms erfolgt über die Konsole die Eingabe "1", gefolgt von *Enter*, dann die Eingabe "2", gefolgt von *Enter*, usw.)
- b. Kommentieren Sie **jede** Zeile des Programms sinnvoll. (Geben Sie bei Ihrer Abgabe die Zeilennummer und den zugehörigen Kommentar an.)
- c. Welche mathematische Funktion berechnet das Programm?

## Aufgabe 30: (H) Einfachauswahlaufgabe: MIPS/SPIM

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Nennen Sie dazu in Ihrer Abgabe explizit die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Der MIPS Prozessor besitzt die folgende Architektur:			
(i) RISC	(ii) MISC	(iii) CISC	(iv) Stack
b) Jedes MIPS-Register hat eine feste Breite. Sie beträgt:			
(i) 32 Bit	(ii) 16 Bit	(iii) 8 Bit	(iv) 4 Bit
c) In der MIPS Architektur steht ein Wort für...			
(i) ...die größte adressierbare Informationseinheit.	(ii) ...die Größe einer Speicherzelle.	(iii) ...die maximale Datengröße, die in einem Rechenschritt verarbeitet werden kann.	(iv) ...die kleinste adressierbare Informationseinheit.
d) Wie muss der Assembler-Befehl lauten, wenn der Inhalt von Register <code>\$t1</code> durch den Inhalt von Register <code>\$t2</code> dividiert und das Ergebnis im Zielregister <code>\$t0</code> gespeichert werden soll?			
(i) <code>div \$t1, \$t0, \$t2</code>	(ii) <code>div \$t2, \$t1, \$t0</code>	(iii) <code>mul \$t2, \$t1, \$t0</code>	(iv) <code>div \$t0, \$t1, \$t2</code>
e) Gegeben sei folgende Zeile in SPIM Code: <code>var: .word 10, 11, 12, 13</code> Welcher Befehl lädt den Wert 11 in das Register <code>\$t0</code> ?			
(i) <code>lw var, \$t0+4</code>	(ii) <code>la \$t0, var+4</code>	(iii) <code>lw \$t0, var</code>	(iv) <code>lw \$t0, var+4</code>

Überblick über die wichtigsten SPIM Assemblerbefehle		
Befehl	Argumente	Wirkung
add	Rd, Rs1, Rs2	$Rd := Rs1 + Rs2$ (mit Überlauf)
sub	Rd, Rs1, Rs2	$Rd := Rs1 - Rs2$ (mit Überlauf)
addu	Rd, Rs1, Rs2	$Rd := Rs1 + Rs2$ (ohne Überlauf)
subu	Rd, Rs1, Rs2	$Rd := Rs1 - Rs2$ (ohne Überlauf)
addi	Rd, Rs1, Imm	$Rd := Rs1 + Imm$
addiu	Rd, Rs1, Imm	$Rd := Rs1 + Imm$ (ohne Überlauf)
div	Rd, Rs1, Rs2	$Rd := Rs1 \text{ DIV } Rs2$
rem	Rd, Rs1, Rs2	$Rd := Rs1 \text{ MOD } Rs2$
mul	Rd, Rs1, Rs2	$Rd := Rs1 \times Rs2$
b	label	unbedingter Sprung nach label
j	label	unbedingter Sprung nach label
jal	label	unbed. Sprung nach label, Adresse des nächsten Befehls in \$ra
jr	Rs	unbedingter Sprung an die Adresse in Rs
beq	Rs1, Rs2, label	Sprung, falls $Rs1 = Rs2$
beqz	Rs, label	Sprung, falls $Rs = 0$
bne	Rs1, Rs2, label	Sprung, falls $Rs1 \neq Rs2$
bnez	Rs1, label	Sprung, falls $Rs1 \neq 0$
bge	Rs1, Rs2, label	Sprung, falls $Rs1 \geq Rs2$
bgeu	Rs1, Rs2, label	Sprung, falls $Rs1 \geq Rs2$
bgez	Rs, label	Sprung, falls $Rs \geq 0$
bgt	Rs1, Rs2, label	Sprung, falls $Rs1 > Rs2$
bgtu	Rs1, Rs2, label	Sprung, falls $Rs1 > Rs2$
bgtz	Rs, label	Sprung, falls $Rs > 0$
ble	Rs1, Rs2, label	Sprung, falls $Rs1 \leq Rs2$
bleu	Rs1, Rs2, label	Sprung, falls $Rs1 \leq Rs2$
blez	Rs, label	Sprung, falls $Rs \leq 0$
blt	Rs1, Rs2, label	Sprung, falls $Rs1 < Rs2$
bltu	Rs1, Rs2, label	Sprung, falls $Rs1 < Rs2$
bltz	Rs, label	Sprung, falls $Rs < 0$
not	Rd, Rs1	$Rd := \neg Rs1$ (bitweise Negation)
and	Rd, Rs1, Rs2	$Rd := Rs1 \& Rs2$ (bitweises UND)
or	Rd, Rs1, Rs2	$Rd := Rs1   Rs2$ (bitweises ODER)
syscall		führt Systemfunktion aus
move	Rd, Rs	$Rd := Rs$
la	Rd, label	Adresse des Labels wird in Rd geladen
lb	Rd, Adr	$Rd := \text{MEM}[\text{Adr}]$
lw	Rd, Adr	$Rd := \text{MEM}[\text{Adr}]$
li	Rd, Imm	$Rd := \text{Imm}$
sw	Rs, Adr	$\text{MEM}[\text{Adr}] := Rs$ (Speichere ein Wort)
sh	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 2^{16} := Rs$ (Speichere ein Halbwort)
sb	Rs, Adr	$\text{MEM}[\text{Adr}] \text{ MOD } 256 := Rs$ (Speichere ein Byte)

Funktion	Code in \$v0	Funktion	Code in \$v0
print_int	1	read_float	6
print_float	2	read_double	7
print_double	3	read_string	8
print_string	4	sbrk	9
read_int	5	exit	10