Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Rechnerarchitektur im Sommersemester 2019
Übungsblatt 5

Abgabetermin: 03.06.2019, 12:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 27. – 31. Mai 2019
Besprechung der H-Aufgaben in den Tutorien vom 03. – 07. Juni 2019

Aufgabe 24: (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 25: (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 26: (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.

- 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.
- 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 "6, 7, 8, 9, 0"(d.h. nach Start des Programms erfolgt über die Konsole die Eingabe "6", gefolgt von *Enter*, dann die Eingabe "7", gefolgt von *Enter*, usw.)?

b. Die folgenden Kommentare beschreiben Teile des Programms `simple.s`. Ordnen Sie den Kommentaren jeweils die minimale Anzahl an Codezeilen zu, die benötigt werden, um das beschriebene Verhalten im Code darzustellen, und geben Sie die Zeilennummer(n) dieser Zeile(n) an!

   i) `str1` wird auf der Konsole ausgegeben.

   ii) Es wird eine Zahl von der Konsole eingelesen.

   iii) Das Programm wird beendet.

   iv) Eine Zählvariable wird um den Wert 1 erhöht.

c. In welchem Wertebereich müssen sich die eingegebenen Zahlen befinden, damit keine Fehlerbehandlung stattfindet (= damit das Label *error* nicht angesprungen wird).

d. Welche mathematische Funktion berechnet das Programm?

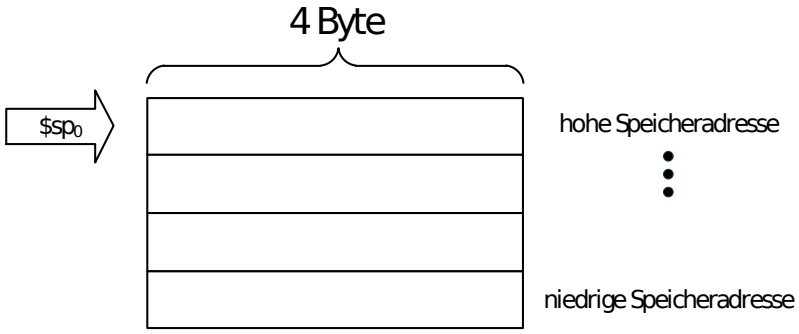
Aufgabe 27: (H) Interaktion mit dem Stack

(5 Pkt.)

Sei folgendes MIPS-Codefragment gegeben:

```
1          li $t0, 30
2          li $t1, 5
3          li $t2, 15
4          li $t3, 10
5          addi $sp, $sp, -12
6          sw $t2, 12($sp)
7          sw $t3, 12($sp)
8          sw $t0, 8($sp)
9          sw $t1, 4($sp)
```

Skizzieren Sie in folgenden Vorlage den Inhalt des Stacks nachdem alle Programmzeilen ausgeführt wurden. Kennzeichnen Sie auch die neue Position des Stackpointers!



Aufgabe 28: (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) MISC	(ii) RISC	(iii) CISC	(iv) Stack
b) Jedes MIPS-Register hat eine feste Breite. Sie beträgt:			
(i) 4 Bit	(ii) 8 Bit	(iii) 16 Bit	(iv) 32 Bit
c) In der MIPS Architektur steht ein Wort für...			
(i) ...die maximale Datengröße, die in einem Rechenschritt verarbeitet werden kann.	(ii) ...die Größe einer Speicherzelle.	(iii) ...die größte adressierbare Informationseinheit.	(iv) ...die kleinste adressierbare Informationseinheit.
d) Wie muss der Assembler-Befehl lauten, wenn der Inhalt von Register `$t1` durch den Inhalt von Register `$t2` dividiert und das Ergebnis im Zielregister `$t0` gespeichert werden soll?			
(i) `div $t1,$t0,$t2`	(ii) `div $t0,$t1,$t2`	(iii) `mul $t2,$t1,$t0`	(iv) `div $t2,$t1,$t0`
e) Gegeben sei folgende Zeile in SPIM Code: `var: .word 10, 11, 12, 13` Welcher Befehl lädt den Wert 12 in das Register $t0?			
(i) `lw $t0, var+8`	(ii) `la $t0, var+4`	(iii) `lw $t0, var`	(iv) `lw $t0, var+4`

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 DIV Rs2
rem	Rd, Rs1, Rs2	Rd := Rs1 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 := MEM[Adr]
lw	Rd, Adr	Rd := MEM[Adr]
li	Rd, Imm	Rd := Imm
sw	Rs, Adr	MEM[Adr] := Rs (Speichere ein Wort)
sh	Rs, Adr	MEM[Adr] MOD 2^{16} := Rs (Speichere ein Halbwort)
sb	Rs, Adr	MEM[Adr] 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