

## Übungsblatt 3

### Rechnerarchitektur im Sommersemester 2023

Zu dem Modul K

**Abgabetermin:** 14.05.23, 18:00 Uhr  
**Besprechung:** 15.05.23 bis 19.05.23

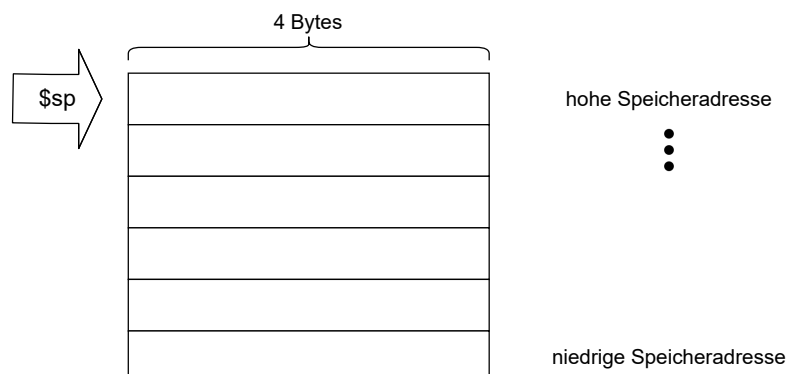
#### Aufgabe Ü1: Interaktion mit dem Stack

(5 Pkt.)

Sei folgendes MIPS-Codefragment gegeben, welches 3 Zahlen auf dem Stack speichern soll:

```
1      li $t0, 5
2      li $t1, 10
3      li $t2, 35
4      addi $sp, $sp, -20
5      sw $t1, 4($sp)
6      sw $t0, 12($sp)
7      sw $t2, 8($sp)
```

- Skizzieren Sie in folgenden Vorlage den Inhalt des Stacks, nachdem alle Programmzeilen ausgeführt wurden. Kennzeichnen Sie auch die neue Position des Stackpointers!
- Welches Problem tritt hier auf?
- Wie könnte man dieses Problem verhindern? Geben Sie die Zeile und ihre Änderung an.



## Aufgabe Ü2: Assemblerprogrammierung unter SPIM

(12 Pkt.)

Laden Sie sich das Programm *simple-counter.s* von der Vorlesungs-Webseite herunter und beantworten Sie die folgenden Fragen zum Thema Assemblerprogrammierung des MIPS-Prozessors.

- a. Ordnen Sie dem Programm *simple-counter.s* jeder Zeile, die mit `# Kommentar <nr> Nr.:` versehen ist, den jeweils sinnvollsten der folgenden Kommentare zu. Ein Kommentar kann auch mehrfach benötigt werden. Schreiben Sie dazu eine Auflistung mit der Kommentarnummer und der Zuordnung zu jeweils einem der folgenden möglichen Kommentare (bspw.: Kommentar 13 Nr.: iii)

- (i) Schaffe Platz auf Stack
- (ii) `while i > 0`
- (iii) Übergebe Argument
- (iv) Sichere i auf Stack
- (v) Lade i vom Stack
- (vi) `i := 10`
- (vii) Sichere übergebenes Argument
- (viii) `i := i - 1`
- (ix) Setze Stackgröße zurück

- b. Geben Sie die letzten drei Zeilen an, die das Programm aus Teilaufgabe a) auf der Konsole ausgibt.

## Aufgabe Ü3: Assemblerprogrammierung unter SPIM

(6 Pkt.)

Im Folgenden soll ein MIPS-Assembler Programm vervollständigt werden, welches als Nutzereingabe eine 8-stellige Binärzahl von der Konsole entgegennimmt, sie in ihre dezimale Repräsentation umwandelt und diese auf der Konsole ausgibt. Die Eingabe wird vom Programm als String entgegengenommen. Danach soll in einer Schleife über die Zeichen diese Strings iteriert und geprüft werden, ob es sich beim aktuellen Zeichen um eine „0“ oder eine „1“ handelt und die entsprechende Wertigkeit aufsummiert werden. Zudem muss der Fall behandelt werden, dass das Ende des Strings, welches durch ein Byte mit dem Zahlenwert 0 markiert ist, erreicht wurde. Beachten Sie, dass der String nach 8 von der Konsole gelesenen Zeichen automatisch übernommen und Null-terminiert wird. Dementsprechend ist kein Zeilenumbruch enthalten.

Laden Sie sich die *binarytodecimal.s* von der Homepage herunter und ergänzen Sie den dort angegebenen Coderahmen um insgesamt **6 Zeilen Code**, sodass das Programm wie beschrieben funktioniert. Tragen Sie Ihre Lösung nur zwischen den mit `# Ihre Loesung:` markierten Stellen direkt in den Coderahmen der heruntergeladenen Datei ein, der restliche Code soll unverändert bleiben.

**Hinweis:** Eine korrekte Lösung mit mehr oder weniger als 6 Codezeilen ist ebenso zulässig.

**Aufgabe Ü4: Einfachauswahlaufgabe: MIPS/SPIM**

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Jedes MIPS-Register hat eine feste Breite. Sie beträgt:			
(i) 16 Bit	(ii) 32 Bit	(iii) 64 Bit	(iv) 8 Bit
b) Der MIPS Prozessor besitzt die folgende Architektur:			
(i) MISC	(ii) Stack	(iii) RISC	(iv) CISC
c) Welche Aussage bei SPIM ist falsch? Der Stack...			
(i) ...arbeitet nach dem FIFO Prinzip	(ii) ...arbeitet nach dem LIFO Prinzip	(iii) ...wächst in Richtung der Speicheradresse 0	(iv) ...hat eine variable Größe
d) In der MIPS Architektur steht ein Wort für...			
(i) ...die größte adressierbare Informationseinheit.	(ii) ...die maximale Datengröße, die in einem Rechenschritt verarbeitet werden kann.	(iii) ...die kleinste adressierbare Informationseinheit.	(iv) ...die Größe einer Speicherzelle.
e) Gegeben sei folgende Zeile in SPIM Code: <code>var: .word 10, 11, 12, 13</code> Welcher Befehl lädt den Wert 13 in das Register \$t0?			
(i) <code>lw var, \$t0+4</code>	(ii) <code>la \$t0, var</code>	(iii) <code>lw \$t0, var+12</code>	(iv) <code>lw \$t0, var+8</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	$Rd := Imm$
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