Freie Universität Berlin

Übung 6

Rechnerarchitektur
Wintersemester 17/18
Prof. Dr.-Ing. Jochen Schiller

Ausgabe 24.11.2017
Abgabe 08.12.2017, 10:15 (s.t.)

Bitte beachten Sie die allgemeinen Hinweise auf Übungszettel 1

Aufgabe 1: Macrobefehle

Im Anhang finden Sie einen Ausschnitt aus dem MIPS Befehlssatz. Dieser Befehlssatz ist turingvollständig, aber etwas klein. Sie sollen ihn um folgende Macro-Befehle erweitern:

```
mov $t, $s           # Kopiert den Inhalt von $s nach $t
push $t              # Kopiert den Wert von $t auf den Stack
pop $t               # Zieht den obersten Wert vom Stack in $t
mult $d, $s, $t      # $d = $s * $t ($d = least 32bit des Ergebnis)
div $d, $s, $t       # $d = $s / $t
mod $d, $s, $t       # $d = $s % $t
not $s               # $s = ~$s
clear $s             # $s = 0
ror $s, C            # Rotiert C viele Bits rechtsherum in $s
rol $s, C            # Rotiert C viele Bits linksherum in $s
bgt $rs, $rt, label # if($rs > $rt) goto label
ble $rs, $rt, label # if($rs <= $rt) goto label
```

Hinweis: Der MIPS Stack ist empty descending. Alle MIPS-Register sind 32bit groß.

Wann bzw. von wem werden in einer RISC Architektur Macrobefehle in Microbefehle übersetzt? Wann bzw. von wem in einer CISC Architektur (wenn überhaupt)?

Aufgabe 2: Typkonvertierung

Implementieren Sie eine „String to Integer“ (*strToInt*) **oder** eine „Integer to String“ (*intToStr*) Funktion. Diese sollen zwischen der ASCII Zeichenkette der Form ”1234” und ihrer entsprechenden Integer Zahlendarstellung *1234* konvertieren.

Die Funktionen sollen folgende Signatur haben:

```
int64_t strToInt(const char* str, uint8_t base);

size_t intToStr(int64_t val, uint8_t base, char* str, size_t len);
```

Parameter:

str - Speicherbereich, der die Zeichenkette aus Zahlen vorhält / in den ASCII-Ziffern geschrieben werden

base - Basis in der die Zeichenkette angegeben ist / werden soll

val - Integer der umgewandelt werden soll

len - maximale Länge die der String haben darf

Rückgabewert: der passende Integerwert bzw. die Anzahl der geschriebenen Zeichen.

Hinweis: Die Funktion strToInt *soll terminieren, sobald ein Zeichen erreicht wird, dass nicht umgewandelt werden kann. Bei angegebenen Basen $b <= 1$ oder $b > 36$ soll die Funktion nichts tun. Sie dürfen davon ausgehen, dass der Eingabe-String nullterminiert ist. Sorgen Sie in jedem Fall dafür, dass der Ergebnis-String nullterminiert ist.*

Anhang

Instructions

Name/Syntax	Semantik	Bemerkung
`add $d, $s, $t`	`$d = $s + $t`	
`sub $d, $s, $t`	`$d = $s - $t`	
`addi $d, $s, C`	`$d = $s + C`	Addiert Konstante
`subi $d, $s, C`	`$d = $s - C`	Subtrahiert Konstante
`mult $s, $t`	`HI:LO = $s * $t`	”HI” und ”LO” sind spezielle Speicherplätze
`multu $s, $t`	`HI:LO = $s * $t`	Wie ”mult” nur vorzeichenlos.
`div $s, $t`	`LO = $s / $t, HI = $s % $t`	”LO” enthält Ergebnis, ”HI” ganzzahligen Rest
`divu $s, $t`	`LO = $s / $t, HI = $s % $t`	Wie ”div” nur vorzeichenlos.
`lw $t, C($s)`	`$t = Memory[$s + C]`	
`sw $t, C($s)`	`Memory[$s + C] = $t`	
`mfhi $d`	`$d = HI`	
`mflo $d`	`$d = LO`	
`and $d, $s, $t`	`$d = $s & $t`	Bitweises UND
`andi $d, $s, C`	`$d = $s & C`	
`or $d, $s, $t`	`$d = $s | $t`	Bitweises OR
`ori $d, $s, C`	`$d = $s | C`	
`xor $d, $s, $t`	`$d = $s ^ $t`	Bitweises XOR
`xori $d, $s, C`	`$d = $s ^ C`	
`nor $d, $s, $t`	`$d = ~($s | $t)`	”~” ist der Bitwise Not-Operator in C
`slt $d, $s, $t`	`$d = ($s < $t)`	$d wird auf 1 gesetzt wenn $s kleiner als $t ist
`sltu $d, $s, $t`	`$d = ($s < $t)`	wie ’slt’ nur Vorzeichenlos
`slti $d, $s, C`	`$d = ($s < C)`	
`sllv $d, $t, $s`	`$d = $t << $s`	Bits in $t werden um $s Stellen nach links geschoben
`srlv $d, $t, $s`	`$d = $t >> $s`	
`srav $d, $t, $s`	`$d = $t >> $s`	Wie ”srlv” nur mit Vorzeichenbit
`sll $d, $t, C`	`$d = $t << C`	
`srl $d, $t, C`	`$d = $t >> C`	
`sra $d, $t, C`	`$d = $t >> C`	Wie ”srl” nur mit Vorzeichenbit
`beq $s, $t, C`	`if($s == $t) goto C`	
`bne $s, $t, C`	`if($s != $t) goto C`	
`j C`	`$gp = C`	
`j $s`	`$gp = $s`	
`jal C`	`$ra = $gp + 4; $gp = C`	

Registers

Register	Nutzung
`$zero`	Konstante 0
`$at`	Assembler temporary
`$v0-$v1`	Funktionsrückgabewerte
`$a0-$a3`	Funktionsargumente
`$t0-$t9`	Zwischenspeicher
`$s0-$s7`	Nicht flüchtiger Zwischenspeicher
`$k0-$k1`	OS-Register
`$gp`	Instruction Pointer
`$sp`	Stack Pointer
`$fp`	Frame Pointer
`$ra`	Return Address