Übung zur Vorlesung Technische Grundlagen der Informatik

Prof. Dr. Andreas Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 09/10
Übungsblatt 12 - Lösungsvorschlag

Aufgabe 12.1 Little Endian / Big Endian

Schreiben Sie eine Funktion in MIPS-Assembler, die zehn 32-Bit Integerwerte, die im Little-Endian-Format gespeichert sind ins Big-Endian-Format konvertiert. Die Werte stehen in einem Array dessen Basisadresse in $a0 steht.

```
little2big:
      addi $t5, $zero, 10  #i=10
loop: lb $t0, 0($a0)       #Lade Byte 0
      lb $t1, 1($a0)       #Lade Byte 1
      lb $t2, 2($a0)       #Lade Byte 2
      lb $t3, 3($a0)       #Lade Byte 3
      sb $t3, 0($a0)       #Schreibe Byte 0
      sb $t2, 1($a0)       #Schreibe Byte 1
      sb $t1, 2($a0)       #Schreibe Byte 2
      sb $t0, 3($a0)       #Schreibe Byte 3
      addi $a0, $a0, 4     #Adresse erhöhen
      addi $t5, $t5, -1    #counter -1
      beq $t5, $zero, done #Abbruch
      j loop
done:
```

Aufgabe 12.2 Vertauschen von Registerinhalten

Geben Sie MIPS-Code-Fragmente an, die die Registerinhalte von $s0 und $s1 unter den jeweils angegebenen Bedingungen vertauschen. Verwenden Sie dabei jeweils insgesamt so wenig Speicher wie möglich.

a) Neben den zu vertauschenden Registern darf nur das Register $t0 verwendet werden. Es darf nur der Befehl `or` verwendet werden.

```
or $t0, $s0, $s0
or $s0, $s1, $s1
or $s1, $t0, $t0
```

b) Nur die Befehle `add` und `sub` dürfen verwendet werden. Außer $s0 und $s1 dürfen keine Register verwendet werden.

```
sub $s0, $s0, $s1
add $s1, $s0, $s1
sub $s0, $s1, $s0
```

Hier kann ein Überlauf entstehen, dessen Behandlung jedoch nicht Gegenstand dieser Aufgabe ist.

c) Nur der Befehl `xor` darf verwendet werden. Außer $s0 und $s1 dürfen keine Register verwendet werden.

```
xor $s0, $s0, $s1
xor $s1, $s0, $s1
xor $s0, $s0, $s1
```

d) Es darf neben den beiden zu vertauschenden Registern nur noch $sp verändert werden.

```
addi $sp, $sp, -4
sw $s1, 0($sp)
move $s1, $s0
lw $s0, 0($sp)
addi $sp, $sp, 4
```

Aufgabe 12.3 Befehlserweiterung

Für spezielle Anwendungen (bspw. in eingebetteten Systemen) setzt man häufig Spezialprozessoren ein, die bestimmte Aufgaben besonders gut bewältigen können. So kommen oft Kryptoprozessoren zum Einsatz, die auf die Berechnung kryptographischer Funktionen spezialisiert sind. Um einen solchen Kryptoprozessor zu bauen, sollen Sie eine MIPS-Architektur als Grundlage nehmen und um die folgenden beiden Befehle zum schnellen Exponentieren erweitern:

- pow $R_d, R_s, R_t : R_d = R_s^{R_t}$
- powi $R_d, R_s, imm : R_d = R_s^{imm}$

Als Opcode können Sie 110101 für pow und 110111 für powi verwenden.

a) Welches MIPS-Befehlsformat würden Sie jeweils verwenden?
Für pow empfiehlt sich das R-Format, für powi das I-Format.

b) Geben Sie die Bitdarstellung der folgenden beiden Befehle an:
pow $s0, $t0, $t1
Die Felder shamt und funct werden hier nicht verwendet und sind daher auf 0 gesetzt.
op rs rt rd shamt funct
110101 01000 01001 10000 00000 000000
powi $t0, $t1, 1111
op rs rt adress
110111 01001 01000 0000010001010111

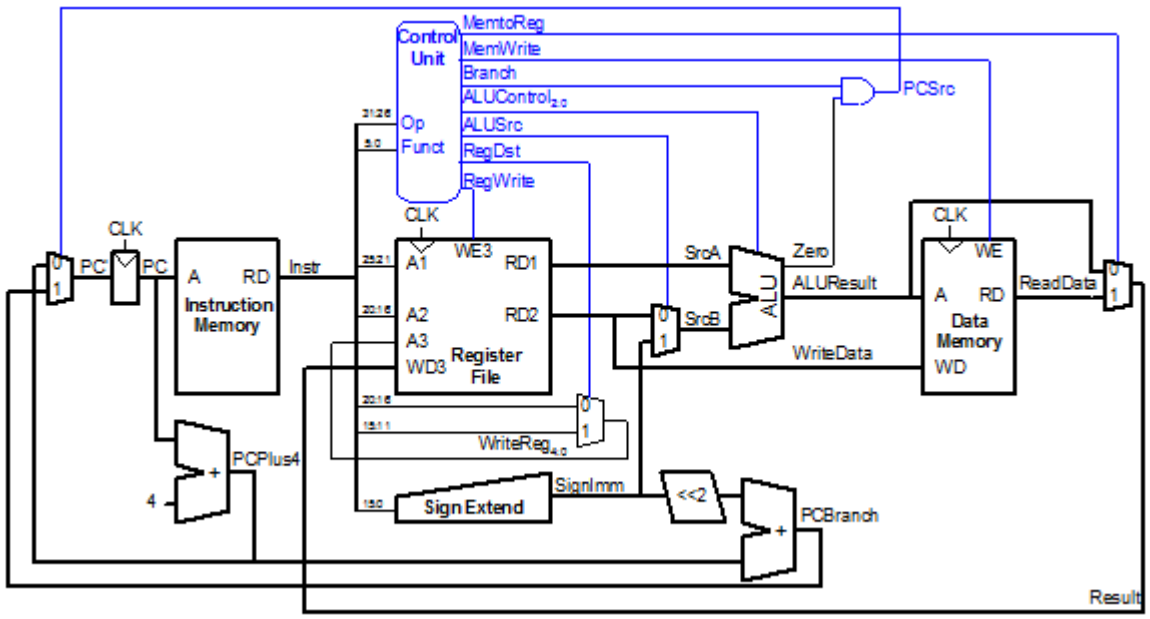
Aufgabe 12.4 1en zählen

Schreiben Sie ein MIPS-Programm, dass die Anzahl von 1en in einem 32Bit-Wort zählt. Das Wort steht im Register $a0. Dieses Register darf nicht verändert werden!

```
      add $v0 $zero $zero   #Ergebnis mit 0 initialisieren
      add $t0 $a0 $zero     #Register kopieren
loop: beq $t0, $zero, done  #Ende, wenn Register = 0
      andi $t1, $t0, 0x1    #Bit testen
      beq $t1, $zero, shift #Bit gesetzt?
      addi $v0, $v0, 1      #Ergebnis + 1
shift:srl $t0, $t0, 1       #shiften
      j loop
done:
```

Aufgabe 12.5 MIPS-Eintaktimplementierung

Betrachten Sie die folgende Implementierung des Eintakt-MIPS-Prozessors:



Welche Komponenten werden bei der Ausführung des Befehls add $t0 $t1 $t2 verwendet? Wie sind die Steuersignale belegt?

Zuerst wird der Befehl aus dem Befehlsspeicher geladen. Das Steuerwerk erhält die Felder `OP` und `Funct` und generiert daraus die zur Durchführung des Befehls benötigten Steuersignale. Die Nummern der Register $t1 und $t2 werden an das Registerfeld angelegt und die Inhalte ausgelesen. Sie werden an die ALU weitergeleitet (`ALUSrc` = 0) und dort addiert (`ALUControl` = 010). Das Ergebnis wird an das Registerfeld zurückgeleitet (`MemtoReg` = 0). `MemWrite` muss 0 sein, da nicht in den Datenspeicher geschrieben wird. Die Adresse des Zielregisters $t0 (`RegDst` = 1) liegt am Registerfeld an und durch Setzen von `RegWrite` = 1 wird das Ergebnis geschrieben. Der PC wird um 4 erhöht (`Branch` = 0). Bei der nächsten Taktflanke wird der nächste Befehl geladen.

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism