TU Dortmund, Fakultät für Informatik
Jansen, Kelter, Munteanu

01. Dezember 2010

Rechnerstrukturen im WS 2010/2011
Übungsblatt 8

Zum Verständnis von Rechnerarchitekturen sind Erfahrungen im Umgang mit einem Assembler unerlässlich. Sie werden auf diesem und den folgenden Übungsblättern eine Reihe von Assembler-Aufgaben finden. Um sie zu lösen, müssen Sie den in der Vorlesung vorgestellten Simulator installieren und mit ihm arbeiten. Sie finden eine vollständige Simulatorsoftware unter:
http://courses.missouristate.edu/KenVollmar/MARS/index.htm (in der Vorlesung empfohlen)

Installieren Sie den Simulator auf Ihrem Rechner.
Wählen Sie unter „File“ „New“ und geben Sie im Editorfeld das nachfolgende Assemblerprogramm ein. Speichern Sie ihre Eingabe als Datei. (z.B. Blatt8A1.asm)

```
.data              # 01 die Zeilenzahl ist zum
x: .word 4711      # 02 Besprechen in der Übung
y: .word 10        # 03
z: .word 0x0a97    # 04
e: .word 0         # 05 Ergebnisvariable
                   # 06
.text              # 07
.globl main        # 08
main:              # 09
      lw  $2,x     # 10
      lw  $3,y     # 11
      lw  $4,z     # 12
      add $2,$2,$3 # 13
      sub $3,$2,$4 # 14
      sw  $3,e     # 15
      li  $2,10    # 16 Programm ordnungsgemaess beenden
      syscall      # 17
```

Lassen Sie das Programm assemblieren (F3). Ihr Programm erscheint nun unter „Execute“ im Textsegment. Es beginnt mit der Zeile: lui $1, 4097 (unter „Basic“). Ihr Eingabetext steht unter „Source“.
Lassen Sie das Programm schrittweise ablaufen (F7). Achten Sie bei jedem Schritt auf Veränderungen in den Registern 2 bis 4 (rechts unter Register „Number“ und „Value“).

Aufgabe 1 (Simulatoroberfläche) (4 Punkte)

a) Nach der Abarbeitung Ihres Programms erscheint unter „Mars Messages“ -- program is finished running (dropped off bottom) --. Welcher Wert steht im Register $3? Geben Sie dieses Ergebnis ebenfalls als Dezimalzahl an.

b) Was berechnet das Programm im Register Reg[3] (bezogen auf die Variablen x,y,z)? Geben Sie sowohl die symbolische Darstellung (Register-Transfer-Notation) als auch die konkreten Werte an.

c) Die erste Programmzeile im Simulator lautet: lui $1, 4097 (unter „Basic“). Woher kommt die 4097?

d) Wo findet man nach Abarbeitung von Zeile 15 das Ergebnis im Speicher? Geben Sie den Bereich der Simulatoroberfläche und sowohl die genaue Speicheradresse als auch deren Inhalt im Format `0x........` an. Was bedeutet der Präfix `0x`?

Aufgabe 2 (Laden von Konstanten) (4 Punkte)

Der Assembler bietet für das Laden einer Konstanten k in ein Register r den Befehl `li` an. Wie setzt der Assembler diesen Pseudobefehl in existierende MIPS-Befehle für folgende Fälle um:

a) $k \bmod 2^{16} = 0$
b) $k \leq 2^{16} - 1$
c) alle anderen Konstanten.

Geben Sie jeweils das allgemeine Prinzip als Mips-Assemblerbefehl an. Sie können auch jeweils ein Zahlenbeispiel angeben.

Aufgabe 3 (Multiplikation) (4 Punkte)

Schreiben Sie ein Assemblerprogramm zum Testen der Multiplikationsvarianten mit den Befehlen mul mult multu und mulo nach dem folgenden Schema:
Legen Sie in zwei Speicherzellen (wert1 und wert2 im Bereich .data) die Konstanten +1 und –1 ab und multiplizieren Sie die Werte zunächst mit mult und dann mit multu. Anschließend berechnen Sie das Produkt der Zahlen 1035 und 4721467 mit den Befehlen mul und mulo.

a) Geben Sie die Ergebnisse der Multiplikationen tabellarisch an (Register Hi und Lo und ggf. Reg[4]). (Das Reg[4] soll dabei, wenn benötigt, als Zielregister benutzt werden).

Befehl	**Hi**	**Lo**	**$4**
mult			
multu			
mul			
mulo			

b) Welches Ergebnis liefern die Befehle mult und multu dezimal? Warum liefert der Befehl mulo kein Ergebnis?

Aufgabe 4 (Befehlssequenz) (4 Punkte)

Gegeben sei folgende Sequenz von MIPS-Befehlen:

(Geben Sie ab der 2. Zeile nur Veränderungen an)

	Registerinhalte nach Ausführung des Befehls			
	$2	$3	$4	$lo
bei Programmstart	?	?	?	?
li $2,0x04				
ori $3,$0,48				
mul $4,$3,$2				
mfhi $3				
addi $3,$4,0x014E				
and $2,$3,0xFF03				
add $3,$2,$3				
ori $4,$3,0x4701				

Hinweise:
Es wird empfohlen, im Hexsystem zu rechnen. Logische *Immediate*-Befehle nutzen die *zero-extend*-, nicht die *sign-extend*-Funktion. Sie sollten diese Aufgabe ohne Benutzung des Simulators lösen können.

Die Abgaben sollen bis Mittwoch, den 08. Dezember 2010 um 20.00 Uhr in die Briefkästen im Pavillon 6 eingeworfen werden. Bitte Name (bei einem 3er-Team alle), Matrikel- und Gruppennummer oben auf der ersten Seite der Lösungen angeben.