Rheinische Friedrich-Wilhelms Universität Bonn
Institut für Informatik VI, Technische Informatik

Prof. Dr. Joachim K. Anlauf
Oguzhan Sezenlik

4. Übung für die Vorlesung

Rechnerorganisation

Sommersemester 2019

Abgabe: Donnerstag, 2.5.2019; Schicken Sie bitte den Quellcode (als lauffähige `.asm` Quelldateien) für die Programmieraufgabe zusätzlich per E-Mail an Ihren Tutor:
Alexandra Chebotareva: `s6alcheb@uni-bonn.de`
Michel Fischer: `michel-fischer@hotmail.de`

Aufgabe 1. *addi/addiu* 3 P.

1. Worin liegt der genaue Unterschied zwischen der `addi` und der `addiu` Instruktion? Konsultieren Sie hierzu Ihre MIPS-Dokumentation.

2. In einem Programm könnte sowohl `addi` als auch `addiu` bei der Berechnung von MIPS-Speicheradressen (wie z.B. der Erhöhung des Stack-Pointers) verwendet werden. Kann eine der beiden Instruktionen hierbei theoretisch zu Problemen führen – falls ja, zu welchen? Begründen Sie Ihre Antwort.

Aufgabe 2. *Pseudoinstruktionen* 7 P.

1. Was sind Pseudoinstruktionen (wie z.B. `move`, `beqz` und `li`) und zu welchem Zweck gibt es diese beim MIPS-Assembler?

2. Geben Sie für jede der drei nachfolgenden Pseudoinstruktionen jeweils eine Folge von echten MIPS-Instruktionen an, die sie realisiert. Verwenden Sie dabei möglichst wenige Instruktionen.

`move`	`rdest`,	`rsrc`	`#` Move (copy) register `rsrc` to `rdest`
`beqz`	`rsrc`,	`label`	`#` Conditionally branch to the instruction `#` at the `label` if `rsrc` equals 0
`li`	`rdest`,	`imm`	`#` Move the immediate `imm` into register `rdest` *Hinweis: Beachten Sie, dass* **`imm`** *ein Wert mit 16 Bit oder 32 Bit sein kann. Geben Sie eine Lösung für beide Varianten an.*

3. Finden Sie die kürzeste Befehlssequenz in MIPS Assembler, die die Berechnung des Absolutbetrags einer Zweierkomplementzahl ermöglicht. Betrachten Sie dazu die folgende, fiktive Pseudoinstruktion:

```
abs $t1, $t2
```

Diese Instruktion soll bedeuten, dass `$t1` eine Kopie von `$t2` enhält, falls `$t2` positiv war, sonst das Zweierkomplement von `$t2`.

Hinweis: Drei Befehle sind ausreichend. Sie dürfen alle Befehle aus der SPIM-Dokumentation verwenden.

Aufgabe 3. *MIPS-Assembler Progammieraufgabe* 12 P.

1. Schreiben Sie ein MIPS-Assemblerprogramm, das ein beliebiges 32bit-Integer-Array der Größe n als Eingabe bekommt und dieses aufsteigend mit den Zahlen 1 bis n füllt.

   Der auf unserer Webseite zur Verfügung gestellte Assemblercode ("*hauptprogramm.asm*") soll hierbei als Hauptprogramm für Ihren Code dienen.

   Hinweis: Benutzen Sie in Ihrem Code ausschließlich die Register `$a0-$a1` *und* `$t0-$t9!`

   Erklärung: Das Hauptprogramm erwartet zuerst die Tastatureingabe einer Zahl $n > 0$ im Konsolenfenster. Daraufhin reserviert es Speicherplatz auf dem Stack für ein 32 Bit Integer-Array mit exakt n Elementen. Anschließend springt es zum Label "algorithm" – der Ort an dem Ihr selbst geschriebener Assemblercode eingefügt werden soll.

   Die Adresse des ersten Array-Elements wird Ihrem Code in Register `$a1` zur Verfügung gestellt, die Anzahl der Elemente (n) in Register `$a0`. Am Ende des Programms wird das Array als Komma-separierte Liste im Konsolenfenster ausgegeben, wodurch Sie die Korrektheit Ihres Algorithmus testen können.

2. Schreiben Sie ein MIPS-Assemblerprogramm, das den folgenden, in Pseudocode formulierten Algorithmus implementiert:

```
Gegeben: n ∈ ℕ \ {0}, Vektor V ∈ ℤ^n mit den Elementen v_0, ..., v_{n-1}
Lokale Variablen: i, j, m, p ∈ ℤ

Initialisiere alle Elemente von V mit 0;
Setze i := 1;
Setze m := ⌊n/2⌋;
Setze v_m := 1;
Solange i < n − 1 wiederhole
    Setze p := v_m;
    Setze j := m;
    Solange j < n wiederhole
       Falls i ungerade dann
          Setze p := p + v_j;
          Tausche die Werte von p und v_j;
       Falls j < n − 1 und i gerade dann
          Setze v_j := v_j + v_{j+1};
       Setze j := j + 1;
    Setze i := i + 1;
Für jedes Element v_j mit 0 ≤ j < m wiederhole
    Setze v_j := v_{n−1−j};
Ergebnis: V
```

   Verwenden Sie wieder das in der vorigen Teilaufgabe eingeführte Hauptprogramm als Rahmen für Ihren Code.

   Geben Sie die jeweilige Ausgabe an, die Ihr Programm für ein n von 1 bis 10 liefert. Können Sie erkennen, was obiger Algorithmus berechnet?

   Hinweis: Benutzen Sie in Ihrem Code ausschließlich die Register `$a0-$a1` *und* `$t0-$t9!`.