Karlsruhe Institute of Technology
Institut für Anthropomatik und Robotik (IAR)

Rechnerorganisation im WS 2017/18

Frohe Weihnachten und ein erfolgreiches neues Jahr

Prof. Dr. Wolfgang Karl
Haid-und-Neu-Str. 7

Dr.-Ing. Ömer Terlemez
Adenauerring 2, Geb. 50.20

Email: ti@ira.uka.de
Web: http://ti.ira.uka.de

Aufgabe 1 (12 Punkte)

Sie möchten eine MIPS-Umsetzung des bekannten Spiels "Vier gewinnt" programmieren. Es sollen zwei menschliche Spieler auf einem Spielfeld mit sechs Zeilen und sieben Spalten gegeneinander antreten können (für die vollständigen Regeln lesen Sie bitte `http://de.wikipedia.org/wiki/Vier_gewinnt#Regeln`).

Das Spiel soll hierbei automatisch nach jedem Zug das Spielfeld ausgeben, die Gültigkeit eingegebener Züge validieren und eine Gewinnsituation bzw. ein Unentschieden erkennen.

Auf der Homepage der Vorlesung finden Sie das Rahmengerüst für das Spiel zum Download (`blattWeihnachten_vier_gewinnt.s`). Dieses enthält bereits die Ein- und Ausgabe sowie die Datenstruktur zur Speicherung des Spielfelds.

Vervollständigen Sie die folgenden Funktionen der Quelltext, um ein funktionsfähiges Spiel zu erhalten:

1. Implementieren Sie die Funktion `peek`, die in den Registern `$a0` und `$a1` Zeile und Spalte einer Spielfeldposition erhält und den Status des Spielfelds abprüft. [2 P.]

   Im Register `$v0` soll die Nummer des Spielers zurückgeliefert werden, dem der Spielstein auf diesem Feld gehört, bzw. 0 falls das Feld leer ist.

2. Implementieren Sie die Funktion `drop_piece`, die in den Registern `$a0` und `$a1` Spalte und Spielernummer erhält und einen Stein für den entsprechenden Spieler in der Spalte platziert. [4 P.]

   Falls die Funktion erfolgreich ist soll `$v0` auf 0 gesetzt werden. Im Fehlerfall (Spalte bereits vollständig gefüllt) soll dieses Rückgabewert-Register auf 1 gesetzt werden.

3. Implementieren Sie die Funktion `check_fin`, die überprüft, ob einer der Spieler gewonnen hat oder ein Unentschieden vorliegt (alle Felder belegt, ohne dass einer der Spieler gewonnen hat). [6 P.]

   Falls einer der Spieler gewonnen hat soll `$v0` auf dessen Nummer (1 oder 2) gesetzt werden, bei einem Unentschieden auf 3. Wenn das Spiel noch nicht beendet ist soll `$v0` auf 0 gesetzt werden.

Hinweise:

- Ergänzen Sie lediglich die genannten Funktionen und nehmen Sie keine sonstigen Veränderungen am Rahmengerüst vor.

  Zur besseren Strukturierung Ihres Programms können natürlich weitere Hilfsfunktionen definiert und verwendet werden.

- Denken Sie an die MIPS-Aufrufkonventionen, z.B. was das Sichern bestimmter Register in einer Funktion vor deren Veränderung betrifft.

Aufgabe 2 (11 Punkte)

Der Lagerroboter Kobi hat eine wichtige Datendisc verloren. Helfen Sie dem Roboter, diese im vorgegebenen Lagerraum wieder zu finden.

Verwenden Sie als Vorlage für Ihre Implementierung das Programmskelett in `blattWeihnachten_robot.s`.

Der Roboter startet in dem im Programmskelett kodierten Lagerraum an der Position $(x, y) = (1, 1)$. Den Weg versperrende Wände sind durch das ASCII-Zeichen `'#'` kodiert. Die Position der Datendisc ist mit durch `'0'` kodiert.

Kobi kann[1] pro Zeitschritt ein Feld vorwärts fahren, eine Linksdrehung vollziehen, an seiner momentanen Position nach der Datendisc sehen sowie prüfen, ob sich in Blickrichtung auf dem Feld vor ihm eine Wand befindet.

1. Finden Sie die Markierung `MYSTIC CODE` im Code. Beschreiben Sie in kurzen Worten, welche Ihnen bekannten Kontrollstruktur höherer Programmiersprachen hier umgesetzt wird und wie dies auf Assemblerebene erreicht wird. **2 P.**
2. Vervollständigen Sie an der markierten Stelle im Programmskelett die Subroutine `kobi_turnLeft`. **2 P.**
3. Vervollständigen Sie an der markierten Stelle im Programmskelett die Subroutine `kobi_checkDisc`. **3 P.**
4. Implementieren Sie in der `main`-Funktion den Suchalgorithmus nach der Rechte-Hand-Regel[2] wie im Kommentar im Programmskelett als Pseudo-Code vorgegeben. **4 P.**

Aufgabe 3 (3 Punkte)

Betrachten Sie eine 8-stufige Pipeline, die mit einer Taktfrequenz von 100 Mhz getaktet ist. Diese Pipeline führt ein Programmstück ohne Sprungbefehle aus, das aus 100 Befehlen besteht und keine Pipeline Stalls erzeugt.

1. Wie lange dauert die Ausführung dieses Programms? **1 P.**
2. Berechnen Sie den Speedup und den Durchsatz. **2 P.**

[1] In Anlehnung an `http://de.wikipedia.org/wiki/Niki_%E2%80%93_der_Roboter`

[2] `http://en.wikipedia.org/wiki/Maze_solving_algorithm#Wall_follower`