Martin-Luther-Universität Halle-Wittenberg
Fachbereich Mathematik und Informatik
Lehrstuhl für Technische Informatik
Prof. Dr. P. Molitor, S. Wefel

Telefon: ☎ 03 45 / 5 52 47 25
E-Mail: wefel@informatik.uni-halle.de

Halle, 30. Oktober 2003

Rechnerarchitektur und Rechnerorganisation (WS 2003/04)

Übungsserie 3

Aufgabe 3.1. (10 Punkte)

Bei dieser Aufgabe handelt es sich um eine Einführung in die Assembler-Programmierung. In der Vorlesung und den Übungen gehen wir insbesondere auf den MIPS-Prozessor ein, für den wir mittels des SPIM-Pakets einen leistungsfähigen Simulator an der Hand haben.

Ihre erste Aufgabe lautet deshalb: Machen sich mit der Funktionsweise des Simulators vertraut, insbesondere mit den Möglichkeiten zum Laden von Assemblerprogrammen und deren Ausführung, sowie den Möglichkeiten zur Ein- und Ausgabe von Werten über die zugehörige Console.

Nutzen Sie zur Lösung der folgenden und zukünftiger Assembler-Aufgaben neben Ihren Vorlesungsmitschriften die SPIM-Dokumentation von Reinhard Nitzsche; die Sie im Portal zur Vorlesung finden:

http(s)://nirvana.informatik.uni-halle.de/lehre/ws0304/raro/php/index.php

Beachten Sie die dort erläuterten Programmierrichtlinien (Kapitel 2-3), die Ein- und Ausgabe unter SPIM (Kapitel 7) und studieren Sie insbesondere die Anmerkungen zur Verwendung des Kellerspreichers (`Stack`) (Kapitel 12).

a) Schreiben Sie ein Assemblerprogramm, das mehrere Integerwerte einliest, diese in den Speicher schreibt und nach Eingabe eines leeren Wertes (nur `Enter` drücken) die eingegebenen Zahlen in umgekehrter Reihenfolge auf der Console ausgibt.

   Verwenden Sie als Datenstruktur zur Speicherung der Werte einen Stack!

   Dokumentieren Sie Ihr Programm übersichtlich und verständlich. Hierzu bietet sich die Verwendung von Kommentaren im Quelltext an.

   Assemblerprogramme ohne sinnvolle Kommentierung werden in dieser und zukünftigen Übungsserien nicht bewertet!

   Negativ-Beispiel: `add $t1, $t2, $t1   # t1 = t1 + t2`
   Positiv-Beispiel: `add $t1, $t2, $t1   # summe = betrag + summe`

b) Erweitern Sie Ihr Programm, so daß dieses nach der Eingabe zweier Zahlen die Eingabe eines Operationssymbols aus der Menge $\{+, -, *, /\}$ erwartet und anschließend die eingegebene Operation mit den zuvor eingebenen Zahlen ausführt. Das berechnete Ergebnis soll auf der Konsole ausgegeben und ebenfalls auf den Stack abgelegt werden, so daß nach der Eingabe einer weiteren Zahl und eines weiteren Operationssymbols ein neues Ergebnis berechnet wird und ebenfalls wieder auf der Konsole ausgebeben und auf dem Stack abgelegt wird. Demzufolge beendet sich dieses Programm nicht von allein! (Wichtig: das Programm sollte den Sonderfall *Division durch Null* nicht ausführen, also keine Exception verursachen!)

Aufgabe 3.2. (6 Punkte)

Beschreiben Sie mit ausführlicher Erklärung die Adressierungsarten

- immediate,
- direkte und
- indirekte Addressierung.

Nutzen Sie hierzu die Syntax des in SPIM integrierten MIPS-Assemblers. Sehen Sie sich dabei einen Befehl an, welcher eine Speicherzelle ausliest und in ein Register $1 lädt. Zusätzlich gibt es Indexregister $2. Gehen Sie detailliert auf die Besonderheiten beim MIPS-Prozessor ein, z.B. wie groß eingebette Werte in Befehlen sein können.

Aufgabe 3.3. (6 Punkte)

Ein Datenbus mit 8 parallelen Daten-Leitungen soll möglichst viele Informationen (Bytes) in Abhängigkeit eines vorgebenes Taktes als Information vom Sender zum Empfänger übertagen. An dem Bus hängen maximal 15 Empfänger, von denen immer nur einer als Ziel der Übertragung über eine Adresse spezifiziert wird. Es gibt keinen Bus-Arbiter.

- Überlegen Sie sich, wieviele zusätzliche Steuer- bzw. Adressleitungen Sie benötigen, um die Datenübertragung durchzuführen. Wir gehen von einer fehlerfreien Übertragung aus, es werden also keine Leitung(en) für Prüfbits benötigt.
- Skizzieren Sie das Übertragungsprotokoll auf dem von Ihnen entworfenen Bus, entweder als Taktdiagramm oder indem Sie es möglichst detalliert in Textform beschreiben.
- Ist Ihr Bus synchron oder asynchron?