veritas iustitia libertas
Freie Universität Berlin

Rechnerorganisation
SS 2005
Übungsblatt Nr. 1

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin

Ausgabe am 22.04.2005 — Abgabe spätestens 06.05.2005, 12:00 Uhr

Bitte bei der Abgabe nicht vergessen:
alle Namen und Matrikelnummern der Gruppe,
Nummer der Übung/Teilaufgabe und
Datum auf die Abgabe schreiben.

Darauf achten, dass die Lösungen beim richtigen Tutor/der richtigen Tutorin abgegeben werden.
Zu spät abgegebene Lösungen werden nicht mehr berücksichtigt!

1. Aufgabe: Sammeln von Informationen (0 Punkte, aber sinnvoll!)

Bitte schauen Sie regelmäßig auf den WWW-Seiten der Arbeitsgruppe, speziell auf denen zur Vorlesung Rechnerorganisation nach: http://nbi.inf.fu-berlin.de/lehre/05/V_RO/. Nicht nur, dass dort aktuelle Informationen hinterlegt werden, hier findet sich auch zusätzliches Material zur Vorlesung.

2. Aufgabe: Mikroarchitektur, MIC und Java (4 Punkte)

Als Beispiel für eine Mikroarchitektur wird in der Vorlesung Rechnerorganisation MIC zu Grunde gelegt. Diese Architektur wird im Detail im Buch zur Vorlesung von Tanenbaum (ab Seite 257) beschrieben. Es gibt hierzu auch diverse Simulatoren und weitere Information. Folgende Schritte sollten Sie unbedingt durchführen, um sich mit dem System für nachfolgende Übungen vertraut zu machen!

1. Gehen Sie auf die Seite http://www.ontko.com/mic1/ und arbeiten Sie die Beschreibung (FAQs etc.) für den Simulator durch. Installieren Sie den Simulator (läuft auf allen Systemen mit Java) und führen Sie die Beispielprogramme aus. Wenn Sie eine graphische Animation der Mikroarchitektur betrachten wollen, dann finden sich Beispiele im Internet (Suchmaschine, Begriffe „MIC-1“, „Animation“, …).
2. Machen Sie sich mit den Funktionen von Simulator und Assembler vertraut, sodass Sie in der Lage sind, die auf dem nachfolgenden Übungsblatt gestellten Aufgaben angehen zu können.
3. Sie müssen noch nicht die Mikroarchitektur vollständig verstanden haben, um folgende Frage zu beantworten und die zugehörigen Schritte durchzuführen:
    a. Starten Sie den Simulator `mic1sim`, laden Sie als Mikroprogramm `mic1ijvm` und als Makroprogramm `echo2`. Dieses Programm finden Sie als „Assemblerprogramm“ (dazu später mehr) unter http://www.inf.fu-berlin.de/inst/ag-tech/teaching/LehreFUSeiten/SS04/19502-V/echo2.jas. Sie müssen es vor dem Laden noch kompilieren!
    b. Führen Sie 4 Einzelschritte mit dem Simulator durch (Step-Button). Welchen Wert können Sie im Feld „Microinstruction“ ablesen? Drücken Sie dabei keine weiteren Tasten, klicken Sie lediglich mit der Maus auf den Step-Button.
    c. Klicken Sie anschließend auf „Run“ und danach klicken Sie in das leere Feld „Standard out“. Geben Sie dann auf der Tastatur das Wort „Testen“ ein (ohne die „“). Was können Sie ablesen?

Eine kurze Erläuterung zum Was, Warum, Wieso:

- Java-Programme werden von einem Java-Compiler in Bytecode übersetzt. Dieser kann von der JVM ausgeführt werden. Eine kommentierte Version dieses Codes für die vereinfachte JVM (IJVM) von Tanenbaum liegt in den *.jas-Dateien.
- Richtige Java-Compiler erzeugen Code, der deutlich komplexer ist als das, was wir hier in der Vorlesung zum Verständnis benötigen. Es soll hier anhand von vereinfachten Java-Bytecode gezeigt werden, wie dieser wiederum auf einer Mikroarchitektur ausgeführt werden kann.
- Solange man keine Hardware hat, die direkt Java-Bytecode ausführt, benötigt man Mikrocode, welcher den Java-Bytecode interpretiert, in einzelne Mikroinstruktionen umsetzt und dann auf der Mikroarchitektur ausführt. Eine solche Architektur ist MIC von Tanenbaum. Aus diesem Grund müssen Sie auch zunächst ein Mikroprogramm in die Miroarchitektur laden. Danach kommt das Makroprogramm (also unser Java-Bytecode).
- Alles weitere zu den Compilern, Simulatoren etc. wird in den Dokumenten im Netz erläutert!

In der Vorlesung wird das Prinzip der Mikroarchitektur anhand von Java und MIC erläutert. Die eigentliche Assembler-Programmierung werden wir mit Hilfe von Visual Studio .NET und der IA32 (Intel Architecture) durchführen. Der Grund ist recht einfach: Java und der zugehörige Assembler sind recht abstrakt und verwenden beispielsweise keine Register (die JVM ist eine Stack-Maschine). Reale Prozessoren sehen aber typischerweise nicht wie die JVM aus, sondern haben Register und zahlreiche weitere Feinheiten. Diese können nur mit einem „echten“ Assembler angesprochen werden.

3. Aufgabe: Mikroarchitektur (10 Punkte)

1. Warum könnte eine CPU fehlerhaft arbeiten bzw. nicht mehr funktionieren, wenn der Takt zu stark erhöht wird? Warum können auch Fehler auftreten, wenn der Takt zu stark verringert wird?
2. Welche Ereignisse könnten eine Unterbrechung auslösen? Nennen Sie mindestens 4. Kann eine Unterbrechung wieder unterbrochen werden? Warum?
3. Welche Vorteile bieten Mikrobefehle, welche Nachteile? Nennen Sie je zwei.
4. Ein Prozessor will pro Takt verzögerungsfrei ein Register lesen können. Innerhalb welcher Zeit muss daher ein Register seinen Inhalt mindestens an die CPU bei einem 3-GHz-PC liefern? Wie weit kommt ein elektrisches Signal höchstens in dieser Zeit?
5. Wie wird im Allgemeinen ein Vergleich zwischen zwei Zahlen in einem Prozessor durchgeführt (also >, < etc.)?
6. Welche Vorteile bieten Kellerspeicher im Vergleich zu beliebigen Registern? Welche Nachteile? (je zwei)
7. Warum ist die JVM eine Stapelmaschine?

4. Aufgabe: POPTHREE (3 Punkte)

Schreiben Sie einen Mikrocode für die MIC-1, um die JVM-Instruktion `POPTHREE` zu implementieren. Diese Instruktion entfernt drei Wörter von der Oberseite des Stapels.

5. Aufgabe: ADDTHREE (3 Punkte)

Schreiben Sie einen Mikrocode für die MIC-1, um die JVM-Instruktion `ADDTHREE` zu implementieren. Diese Instruktion addiert die obersten drei Wörter des Stapels und legt das Ergebnis wieder auf dem Stapel ab.