Fachhochschule Wiesbaden
Studiengang Medieninformatik
Sommersemester 2012
Prof. Dr. Robert Kaiser
M. Sc. Kai Beckmann

Betriebssysteme und Rechnerarchitekturen

LV 4112

Übungsblatt 2

26.03.12

Aufgabe 2.1 (Modellcomputer MoCo V0.1)

Auf der Webseite der Lehrveranstaltung finden Sie eine Beschreibung der Version 0.1 des MoCo (Modell-Computer), sowie ein Programm, das diesen Computer simuliert. Der Simulator liest eine auf der Kommandozeile angegebene Datei ein und führt sie als Maschinenspracheprogramm aus.

(a) Arbeiten Sie das Dokument durch, installieren Sie den Simulator und erproben Sie ihn anhand des angegebenen einfachen Beispiels. Verwenden Sie zunächst noch nicht den ebenfalls im Dokument beschriebenen Assembler, sondern erzeugen Sie eine ausführbare Datei `prog.exe` z.B. wie angegeben mit einem kleinen C-Hilfsprogramm. Erstellen Sie eine Datei, die das folgende Maschinenprogramm enthält:

```
IN  0, R1
IN  0, R2
ADD R1, R2, R3
OUT R3,0
HALT
```

Hinweis: Dieses Programm entspricht folgender Zahlenfolge:
1, 0, 1, 1, 0, 2, 4, 1, 2, 3, 2, 3, 0, 0

Was tut dieses Programm? Erproben Sie es und überprüfen Sie die Richtigkeit Ihrer Vorhersage.

(b) Erstellen Sie eine Datei mit folgendem Maschinenprogramm:

```
loop:   IN 0,R1
        IN 0,R2
        ADD R1,R2,R3
        OUT R3,0
        SUBN R3,10,R3
        JMPNZ loop
end:    HALT
```

Hinweis: Dieses Programm entspricht der Zahlenfolge:
1, 0, 1, 1, 0, 2, 4, 1, 2, 3, 2, 3, 0, 9, 3, 10, 3, 14, 0, 0

Welches Verhalten erwarten Sie von diesem Programm? Erproben Sie es mit Ihrem MoCo-Simulator. Zeigt es das erwartete Verhalten?

(c) Schauen Sie sich den Quellcode des Simulators an und machen Sie sich mit seiner Arbeitsweise vertraut. Welche Teile müssen Sie wie verändern, um den Befehlssatz des Prozessors zu erweitern?

Aufgabe 2.2 (MoCo-Assembler):

Von nun an soll der MoCo Assembler, `moco-ass` verwendet werden, um Maschinenspracheprogramme in ausführbaren Binärcode zu wandeln.

Hinweis: Beachten Sie, dass der Assembler im Gegensatz zum Simulator bereits den erweiterten Befehlssatz des MoCo-Prozessors unterstützt, d.h. Er braucht im Rahmen dieser Übung nicht verändert zu werden.

(a) Lesen Sie sich die Beschreibung in der Kurzanleitung durch, kompilieren Sie den Assembler und erproben Sie ihn an den Beispielprogrammen aus der vorangegangenen Aufgabe.

(b) Schauen Sie sich den Quellcode des Assemblers an. Es handelt sich hier um einen sogenannten ,,2-Pass" Assembler. Ergründen Sie anhand des Quellcodes, was darunter zu verstehen ist, und wozu diese Technik notwendig ist.

(c) Der Assembler unterstützt auch drei Pseudo-Instruktionen namens DC (define constant), DS (define storage) und DB (define bytestring). DC und DS benötigen eine Zahl als Operand, DB einen Textstring (in Anführungszeichen). Erweitern Sie ein Maschinenprogramm (zum Beispiel eines aus der vorigen Aufgabe) um folgende Zeilen:

```
var:    DC      -1
        DB      "Hello World\n"
        DS      100
stk:
```

Assemblieren Sie das Programm und vergleichen Sie es bezüglich Funktion, Größe und Inhalt mit der vorigen Version. Wozu braucht man diese Pseudo-Instruktionen?

Aufgabe 2.3 (MoCo Version 0.2):

Bisher fehlen dem MoCo Computer noch einige wesentliche Eigenschaften:

1. Er kann mit Ausnahme der Register keine Variablen verarbeiten
2. Er kennt keinen Stack
3. Er kennt keine Unterprogramme

Diese drei Probleme sollen im Rahmen der folgenden Aufgaben beseitigt werden.

(a) Erweitern Sie den Befehlssatz des Prozessors um folgende Befehle:

Name	**Opcode**	**Op1**	**Op2**	**Op3**	**Operation**
LOAD	17	ra	rt	-	Lade rt mit Inhalt der Speicherzelle, auf die ra zeigt
LOADB	18	ra	ro	rt	Lade rt mit Inhalt des Speicherbytes, auf das ra+ro zeigt
STOR	18	rs	ra	-	Speichere rs in die Speicherzelle, auf die ra zeigt
STORB	20	rs	ra	ro	Speichere rs in das Speicherbyte, auf das ra+ro zeigt

Hinweise: *Wort*-Zugriffe auf ungerade Adressen sind unzulässig und müssen -ebenso wie Zugriffe ausserhalb des Speicherbereiches- eine Exception auslösen. Byte-Zugriffe dürfen hingegen auch ungerade Adressen ansprechen.

Erproben Sie die neuen Befehle mit Hilfe eines Programmes, das eine im Speicher befindliche Variable liest, sie verändert und den neuen Wert speichert.

(b) Erweitern Sie den Befehlssatz des Prozessors um folgende Befehle:

Name	**Opcode**	**Op1**	**Op2**	**Op 3**	**Operation**
PUSH	21	rs	-	-	Speichere Inhalt von rs auf Stack
POP	22	rt	-	-	Lade rt vom Stack

Hinweis: Verwenden Sie Register R15 als Stackpointer

Erproben Sie die neuen Befehle mit Hilfe eines Programmes, das einen Registerinhalt auf den Stack speichert, und diesen anschließend wieder von dort lädt.

(c) Erweitern Sie den Befehlssatz des Prozessors um folgende Befehle:

Name	**Opcode**	**Op1**	**Op2**	**Op3**	**Operation**
JSR	23	a	-	-	Rufe Unterprogramm an Stelle a
RET	24	-	-	-	Kehre zum Aufrufer zurück

Erproben Sie die neuen Befehle mit Hilfe eines Programmes, das eine Subroutine aufruft.

(d) Schreiben Sie ein Assemblerprogramm, das rekursiv die Fakultät einer eingegebenen Ganzzahl berechnet. Was geschieht, wenn die Verschachtelungstiefe zu groß wird?