Universität des Saarlandes
FR 6.2 - Informatik
Prof. Dr. W.J. Paul
Dipl.-Ing. Christoph Baumann

7. Übungsblatt Informatik II

(Abgabe: 16.06.2008)

Aufgabe 1: (Incrementer) **(5+5 Punkte)**

Im *nextPC* Schaltkreis wird ein 30-Bit-Incrementer verwendet. Ein n-Bit-Incrementer ist ein Schaltkreis, der die Funktion Inc berechnet.

$$Inc : \{0,1\}^n \to \{0,1\}^n, \qquad Inc(x) = x +_n 1_n$$

Konstruiere **jeweils einen** n-Bit-Incrementer mit:

a) linearen Kosten, d.h. die Kosten sollen in $\Theta(n)$ liegen

b) logarithmischer Tiefe ($D(n) \in \mathcal{O}(\log n)$)

Gib jeweils die genauen Kosten und die Tiefe deiner Konstruktion in einer geschlossenen Form an!
Hinweis: Falls für Funktionen f und g $f \in \Theta(g)$ gilt, so folgt daraus $f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f)$, das heißt, dass f dieselbe Komplexität wie g besitzt und asymptotisch weder schneller noch langsamer, also genauso schnell wächst.

Aufgabe 2: (Speicherkorrektheit) **(10 Punkte)**

Beweise die dritte Zeile des Simulationssatzes der DLX für $x \in \{0,1\}^{30}$:

$$h^{2i+2}.m(x) = c^{i+1}.m_4(x00)$$

Es dürfen alle Definitionen aus der Vorlesung und die Induktionsvoraussetzung $sim(c^i, h^{2i})$ verwendet werden.

Aufgabe 3: (DLX - Assemblerprogrammierung) **(6+7+7 Punkte)**

In den folgenden Aufgaben sollen DLX-Assemblerprogramme geschrieben werden. Zum Bearbeiten stehen alle Instruktionen aus dem Instruktionssatz (s. Webseite) zur Verfügung. Beachte auch die am Ende dieses Übungsblattes aufgeführten Hinweise zu der von uns verwendeten Assembler Syntax und das Beispielprogramm.
Die Assemblerprogramme zu den Aufgaben b) und c) müssen am Computer geschrieben und per Email an deinen Übungsleiter gesendet werden. Alle Programme müssen zudem verständlich kommentiert werden. Programme ohne Kommentare werden nicht bewertet.

a) Der Inhalt der Variable x sei in Register `R5` gespeichert. Schreibe jeweils ein Assemblerprogramm, welches die folgenden Programmfragmente einer fiktiven Hochsprache berechnet! Dabei seien A und B Teilprogramme, die in Assemblerbefehlsketten `code`(A) und `code`(B) der Länge m bzw. n übersetzt werden können.

- `while` $(x > 0)$ {A}
- `if` $(x > 0)$ {A} `else` {B}

b) Schreibe ein Assemblerprogramm, dass die ersten n Fibonacci-Zahlen in den Speicher schreibt! Dabei soll bei Wortadresse 1025 begonnen und diese sowie die darauffolgenden Speicherzellen beschrieben werden. Jede Zahl soll ein Speicherwort (32 Bit) belegen. n steht zu Anfang im Speicher an der Wortadresse 1024.

c) Schreibe ein Assemblerprogramm, dass den größten gemeinsamen Teiler zweier natürlicher Zahlen berechnet! Die beiden Zahlen stehen zu Anfang im Speicher an den Wortadressen 815 und 816. Das Ergebnis soll in das Speicherwort mit der Adresse 817 geschreiben werden.
Tipp: Implementiere den Euklidschen Algorithmus!

Hinweis
Von uns verwendete Assembler Syntax (abhängig vom Instruktionstypen):

I - type	R - type	J - type
mnemonic `RD RS1 imm`	*mnemonic* `RD RS1 RS2`	*mnemonic* `imm`
Ausnahmen:		
`beqz`, `bnez`: *mnemonic* `RS1 imm`		
`jr`, `jalr`: *mnemonic* `RS1`		

Statt *mnemonic* ist der Name des jeweils gewählten Befehls anzugeben. Eine Zeile sollte `PC`, Befehlsname, Register und einen Kommentar (eingeleitet durch "`;`") enthalten.

Beispielprogramm: Das das folgende Codefragment berechnet `while (i>10) do {i=i+1}`:

```
 0:  clr   R0 R0 R0  ; R0=0
 4:  addi  R5 R0 i   ; R5=i
 8:  addi  R6 R0 10  ; R6=10
12:  addi  R7 R0 1   ; R7=1
16:  sls   R8 R5 R6  ; if (i<10) then R8=1
20:  beqz  R8 12     ; if R8=0 goto 32
24:  add   R5 R5 R7  ; R5=R5+1
28:  j     -12       ; goto 16
32:  ...
```

Zum Testen und Debuggen von DLX-Assemblerprogrammen kann Christian Müllers Simple DLX Simulator[1] verwendet werden.

[1] SDS - http://code.google.com/p/simple-dlx-simulator/