Prof. Dr. Christian Steger
Prof. Dr. Reinhard Wilhelm

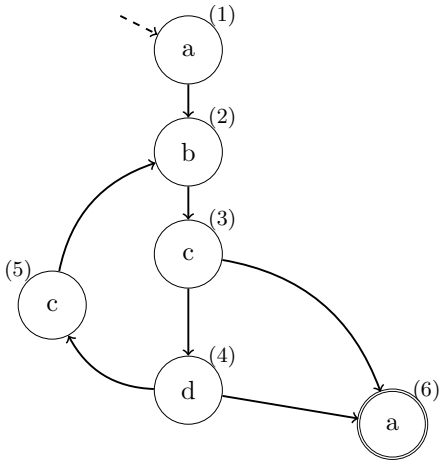
Sebastian Altmeyer, M.Sc.
Michael Gerke, M.Sc.
Dipl.-Inf. Hans-Jörg Peter

Embedded Systems 2010/2011 – Assignment Sheet 10

Due: Tuesday, 1st February 2011, *before* the lecture (i.e., 10:10)
Please indicate your **name**, **matr. number**, **email address**, and which **tutorial** you are planning to attend on your submission. We encourage you to collaborate in **groups** of up to **three** students. Only one submission per group is necessary. However, in the tutorials every group member must be capable to present each solution.

Exercise 1: May/Must LRU Cache Analysis (30 pts.)

Assume an initially empty 4 way fully-associative LRU cache. Perform the *may*- and *must*-cache on the following control-flow graph. Provide the abstract cache-states at each program point.



Exercise 2: Comparison LRU/FIFO/PRLU (20 pts.)

Given an initially empty fully-associative cache with 4 ways. Provide a sequence of accesses such that

(a) LRU outperforms FIFO.

(b) FIFO outperforms LRU.

(c) PLRU outperforms LRU.

(d) PLRU outperforms FIFO.

Note that policy A outperforms policy B iff A exhibits more cache hits than B on the same sequence of accesses.

Exercise 3: Cache-Predictability (30 pts.)

To be on the safe side, one has to assume that the cache-content is completely unknown at the beginning of the program execution. This means, every element may or may-not be cached. Thus, an important cache-predictability metric is the number of cache-misses one has to encounter before one can precisely know the cache-content.

Given a 4-way fully associative cache without any knowledge about the cached elements. For each of the following statements, provide an accesses sequence and an initial cache state to illustrate it:

(a) In case of LRU, one needs exactly 4 accesses until cache content is known.

(b) In case of FIFO one may need up to 8 accesses until the cache content is known.

(c) In case PLRU, even if the number of cache-misses is infinite, one element may survive in the cache without ever being accessed.

Exercise 4: Hardware/Software Partitioning (20 pts.)

A set of function objects $O = \{o_1, \ldots, o_n\}$, $n \in \mathbb{N}$, can either be implemented in hardware (HW) or software (SW). Each object o_i, $1 \leq i \leq n$, has HW costs $c_h(i)$, SW costs $c_s(i)$, HW computation time $d_h(i)$, and SW computation time $d_s(i)$.

(a) Encode the described partitioning problem as an integer programming problem, where cost and computation time are weighted "u USD per second" in the cost function.

(b) Extend your problem such that the total number of function object implementations in HW must not exceed H_{max}.

(c) Extend your problem such that the total costs must not exceed C_{max} and the total computation time must not exceed D_{max}.

(d) Extend your problem such that some pairs of objects must be implemented either both in HW or both in SW. Assume that these pairs are given by a relation $\{(o_1, o'_1), \ldots, (o_m, o'_m)\} \subseteq O \times O$, $m \in \mathbb{N}$.