Saarland University
Department 6.2 - Computer Science
Prof. Dr. W. J. Paul
M. Sc. Christian Müller

Computer Architecture I - WS 07/08

Exercise Sheet 5

Excercise 0: (outline)
In the class we have *formally specified* the DLX processor [MP00]. A processor configuration $c \in C$ was defined as $c = (PC, GPR, m)$. I.e., each configuration consists of a byte addressable memory m, set of write/read-registers GPR and the program counter PC. PC is a pointer to some data in the memory m. This data (also called instruction) is interpreted by the processor as code to execute (this could be a user program, an operating system, etc.). To execute an instruction extracted from the memory, it should be parsed in order to get its semantic (this is done, e.g. with the help of the opcode $I(c)[31:26]$ or function code $I(c)[5:0]$ of each instruction).

GPR (general purpose registers) are needed to store some intermediate data. In a theoretical model (in the specification) there is no need for such registers, since we also could use some range of m for this purpose. However, in a real computer architecture, each memory access is an extremely expensive process and additional on-chip registers of the processor could significantly speed up its computation. Therefore, each real processor has a counterpart for these GPR registers. The most obvious example for the need of GPR registers could be the following situation: imagine, we want to read out the content of two memory addresses, sum them, and write the result back to the memory. Since the processor can make only one access to the memory at a time, we have to load the first value, then to store it in GPR, then we have to load the second value, and to store it in GPR. Afterwards, we sum them and write the result into GPR. Finally, we can write the result from GPR back to the memory.

All these processor computations are modelled as chains of its configurations. In order to get reasonable computations (chains), we require a transition function δ_c, which transform one DLX configuration into another by a certain reasonable way. The most difficult part is to *specify* this transition function δ_c. I.e., we want to describe how one configuration transforms into another. This process obviously depends on the kind of the executed instructions and also was described in the class.

Excercise 1: (DLX specification)

1. Outline differences between a *formal specification* and an *implementation*.
2. In the class we have already specified the result of the function $alures(c)$ which describes some arithmetical/logical result for the given configuration c. Specify the result of the function $shiftres(c)$, which should describe the shifting result for the given configuration c.
   Hint: First, *specify* (do not construct it!) some function $shift\_unit(a, b, cmd)$ which shifts the number a by b bits using one of known to us approaches (lls, lrs, cls, crs, ars, ...) according to the command cmd. The most easiest way to do this – just to describe formally the shift unit presented in the class. Second, combine $shiftres(c)$ and $shift\_unit(a, b, cmd)$ taking all needed arguments from the current configuration c.

Saarland University
Department 6.2 - Computer Science
Prof. Dr. W. J. Paul
M. Sc. Christian Müller

Computer Architecture I - WS 07/08

Exercise Sheet 5

3. Write an assembler program which computes a^b for the given 32-bit numbers a, b if and only if $a \bmod 2 = 0$ and $b \bmod 3 = 0$. Otherwise, your program should return 0. Assume, a and b are stored in the memory at addresses $maddr_a$ and $maddr_b$ respectively. The result should be stored to the address $maddr_r$.

4. Prove the correctness of your program from the previous exercise, showing the chain of DLX configurations, which are computed from c_0 by applying the specification of δ_c from the lecture. For this, you may assume, that your program is stored in the memory at some address $maddr_s$ and assume you are given the first DLX configuration c_0 with $I(c_0) = maddr_s$.

   You can use all known instructions from the lecture (the overview can be found in [MP00], pages 65-66). And, yes, you are allowed to abbreviate similar steps.