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

Computer Architecture I - WS 07/08

Exercise Sheet 8

Excercise 1: (warm up)

1. Assume you are writing an assembler program and want to store a 32-bit constant into one of the GPR registers. How would you do this?[1]
2. What is ghost hardware and what is it used for?
3. Explain the idea of forwarding.
4. In our pipelined processor with the forwarding mechanism we use for some stages additional boolean flags – valid bits. Why do we need them?

Excercise 2: (self-modifying code)
At the moment, the DLX can not execute a self-modifying code correctly for several reasons. However, assume it is already possible. Write an ISA program that writes into each memory cell from address 1024 to address 32767 its address modulo 2^8. That is, write into memory address 1024 the number 0, into memory address 23444 the number 148, etc. Thereby, fulfill the following constraints:

1. You are only allowed to use absolute addresses, i.e. on a memory access the effective address $ea(c)$ has to be computed as $(R0 + imm)$.
2. Your code consists of eight or less instructions (it is possible with 6).

You may assume that your code starts at memory address 0.

Excercise 3: (fast forwarding circuit)
In class, we have seen a forwarding circuit capable of forwarding data from 3 stages. This construction can be generalized to an s-stage forwarding, with $s > 3$, where the data selection is then performed by s cascaded multiplexers. Each multiplexer is controlled by a signal $top.j$. The delay of this realization is in $O(s)$. Construct an alternative circuit, which forwards the requested data from s stages with a delay in $O(log(s))$.

[1]From now on you can test all your assembler programs with SDS; see the webpage news for more information.

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

Computer Architecture I - WS 07/08

Exercise Sheet 8

Excercise 4: (forwarding and hardware interlock: deadlock)
Assume, we have an interlock engine that stalls the stages IF and ID in case of a data hazard, i.e. in case the forwarding engine cannot deliver the right data. However, if the data is available, the forwarding circuit signals a hit in a stage $j \in \{2, 3, 4\}$ by

$$hit[j] = full.j \wedge gprw.j \wedge (Cadr.j = adr) \wedge (adr \neq 0^5)$$

These hit signals are used to generate the top signals. These are then used to compute the data hazard signals $dhazA$ ($dhazB$ analogously) as

$$dhazA = topA.2 \wedge \neg v[2].2 \vee topA.3 \wedge \neg v[3].3$$

In this exercise, you will show that the check whether stage j is full (i. e. $full.j = 1$) is essential for the correctness of the interlock mechanism. Thus, show that if we simplify the hit signal computation to

$$hit[j] = gprw.j \wedge (Cadr.j = adr) \wedge (adr \neq 0^5)$$

then some instructions could also activate the hazard flag, and that the interlock engine could run into a deadlock.