Saarland University
Department 6.2 – Computer Science
Dr. M. A. Hillebrand
Prof. Dr. W. J. Paul

Computer Architecture I – WS 06/07

Exercise Sheet 7 **(due: 18.12.06)**

Exercise 1: (stabilizer circuit for instruction fetch) **(9 points)**

Recall that our memory protocol required the master to keep its input stable during the whole request. If the master does not guarantee this, it is necessary to construct an intermediate circuit, which we call *stabilizer*, that guarantees this condition.

In this exercise, you have to construct a circuit that stabilizes the memory inputs for instruction fetch. These inputs are the fetch signal *imr* and the fetch program counter *pc*.

Construct a circuit which is placed between the processor and the instruction memory with the following inputs and outputs:

- an input $p.pc \in \mathbb{B}^{30}$ which denotes the *pc* address from the processor.
- an input $p.imr \in \mathbb{B}$ which denotes the instruction memory read signal.
- an input $m.ibusy \in \mathbb{B}$ which denotes the busy signal from the memory.
- an output $p.ibusy \in \mathbb{B}$ which denotes the busy signal to the processor.
- an output $m.pc \in \mathbb{B}^{30}$ which denotes the *pc* address to the memory.
- an output $m.imr \in \mathbb{B}$ which denotes the memory read signal to the memory.

Your construction should fulfill the property that once a request to the memory is started ($m.imr = 1$), the outputs $m.pc$ and $m.imr$ are kept stable during the whole request. Furthermore, in case the $p.pc$ input changes during request, there should start a new request to the memory after the first one has finished. The processor should not notice the second request. Hence, if $p.imr = 1$ and $p.ibusy = 0$ then the correct data has to be provided according to the $p.pc$ of the same cycle.

Exercise 2: (program execution) **(7 points)**

Consider the algorithm from the last exercise sheet slightly modified to fulfill the software conventions:

0: $beqz(R1, 16)$
4: $add(R2, R2, R1)$
8: $subi(R1, R1, 1)$
12: nop
16: $j(-20)$
20: nop

Execute the program on the pipelined processor without forwarding and list the content of each register in each state for 10 execution cycles. The datapaths were presented in the lecture from november 27th. Assume the simple stalling engine presented in the lecture from december 4th (ue_k^t always enabled except at the beginning after reset). Hence, there are no busy signals from the memories (IM and DM).

Exercise 3: (stalling engine and scheduling function) **(3+3+3=9 points)**

Consider the construction of the stalling engine in the lecture notes from Lecture 11 (November 22nd) and the update enable signals from class defined as

$$ue_k^t = \neg stall_k^t \wedge full_k^t$$

where

$$stall_k^t = stallIn_k^t \vee genStall_k^t$$

Furthermore, consider the scheduling function as defined in lecture

$$sI(0,0) = 0$$

$$sI(0,t+1) = \begin{cases} sI(0,t)+1 & \text{if } ue_0^t \\ sI(0,t) & \text{if } \neg ue_0^t \end{cases}$$

$$sI(k,t+1) = \begin{cases} sI(k-1,t) & \text{if } ue_0^t \\ sI(k,t) & \text{if } \neg ue_0^t \end{cases} \quad \text{for } k > 0$$

Prove the following properties which we used in the correctness proof of the pipelined implementation:

- $full_k^t \wedge ue_{k-1}^t \Rightarrow ue_k^t$
- $sI(k,t+1) = sI(k,t) +$ if ue_k^t then 1 else 0
- $sI(k,t) = sI(k+1,t) +$ if $full_{k+1}^t$ then 1 else 0