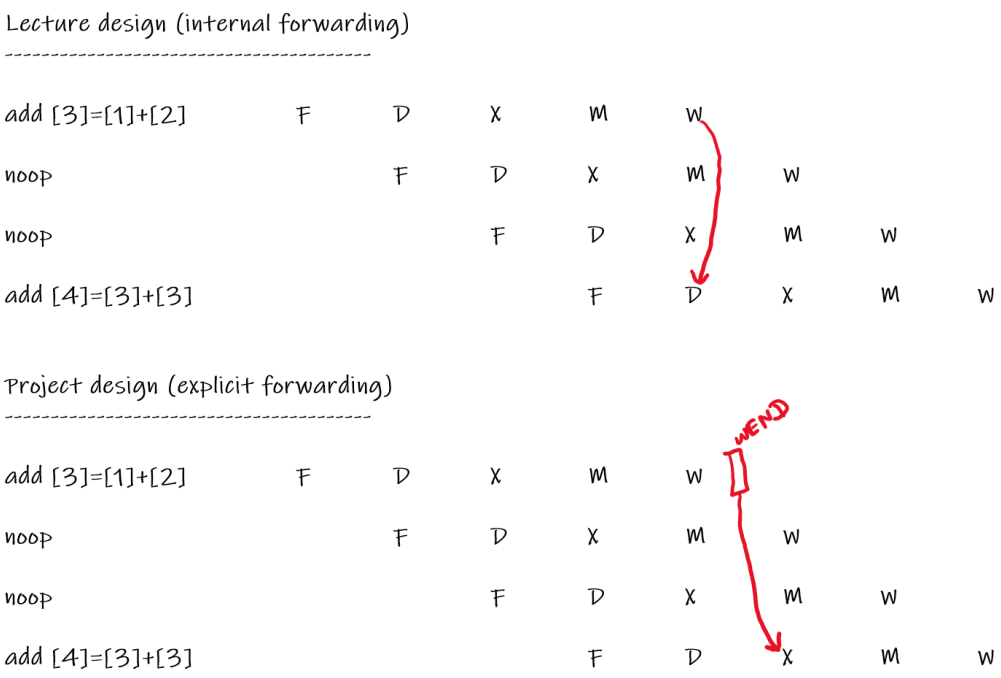
Internal Forwarding

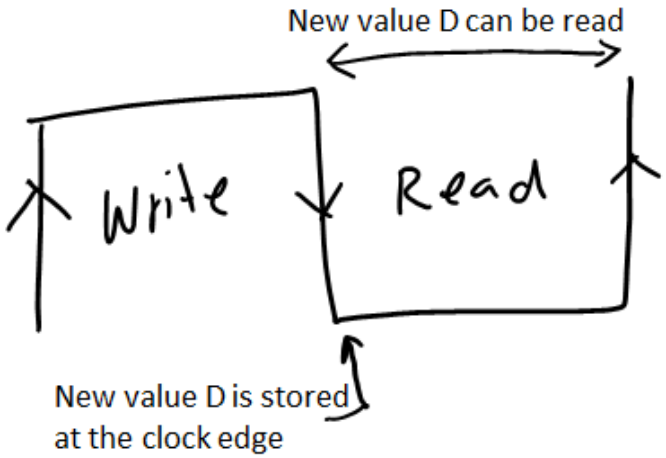
With our lecture register file, we expect to be able to write a new value to a register and to be able to read that new value on the same clock cycle. This is because in lecture we use internal forwarding for our register file. This means when an instruction writes to the register file in the WB stage, this updated value can be read in the same clock cycle by an instruction in the ID stage.

In the project pipeline, we do not have internal forwarding for our register file. This means when an instruction writes to the register file in the WB stage, this updated value cannot be read in the same clock cycle by an instruction in the ID Stage. Consider the following example:



Lecture design 1: Internal forwarding between WB->ID

We assume that it is possible to write data (D) to a register, and read that data D in the same cycle. Edge triggered flip-flops make this feasible. You can write in the first half of a clock-cycle, and the effect of the write takes into effect at the negative clock edge. Then, it is possible to read the newly written data during the second half of that same clock cycle. See the picture below.



Benefit:

For detect and forward pipeline design, we do not need additional hardware to forward data from the write-back to the EX stage. We also do not need to detect a data hazard between the instruction in the ID stage with the instruction in the WB stage. We only need to compare the two source registers of the instruction ID stage with the destination registers of instructions in EX and MEM stage, not WB stage.

For detect and stall pipeline design, we only incur two cycle bubble (as opposed to three) when an instruction is data-dependent on the immediately preceding instruction.

Downside: May increase clock period:

This design impacts the clock period. Note that the write and read to the register file has to be done sequentially, so the clock period has to be long enough to permit that.

Project design 2: No Internal forwarding between WB->ID

We do not assume that a read or a write can be done in half a clock cycle. As a result, data written back to the register file by an instruction in the WB stage cannot be read in the same clock cycle by an instruction in the ID stage.

Benefit: Less impact on clock period:

Register write (from the WB stage) and register reads (from the ID stage) can be done in parallel. So the register latency has lower impact on the clock period compared to the previous design.

Downside:

We need to add a pipeline latch (WEND) to store the intermediate result of the WB stage.

Then, for the detect and forward pipeline design, we need additional hardware to detect a data hazard between the instruction in the WB stage and the instruction in the ID stage. If you go back to slide 40 of lecture 13, we would need two additional comparisons. Plus, the input MUXes in the EX stage need to be extended to forward the data from the WEND register to the EX stage.

For detect and stall pipeline design, we will incur three cycle bubble (as opposed to two) when an instruction is data-dependent on the immediately preceding instruction.

Rationale for considering two designs

You may be wondering why we need to consider two designs in 370. We feel that design 1 is easier to understand first in the lecture. Design 2 is easier to implement and simulate, and therefore is more appropriate for project 3. Understanding two designs also allows one to understand these concepts more deeply. But one could have a fair debate around this choice.