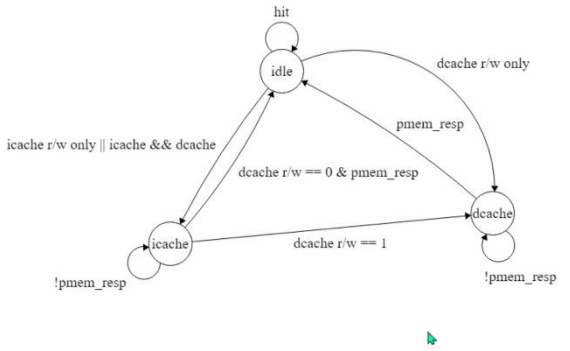
Arbiter, Hazard detection, and Data Forwarding

Anthony Nyugen – alnguyn2, Hemang Nehra – hnehra2

Arbiter Design



In our arbiter design, we have given priority to handle misses by instruction cache first in case data cache miss also arises at the same time. From ideal state, if I-cache issues a miss or both do, then we first pass through the I-cache state. Here memory is accessed to fetch the right instruction. If pmem_resp is set and d-cache doesn't have a request, then move to idle state. Else, serve the d-cache's request. Wait till pmem_resp goes high, and then move back to idle.

Detecting Data Hazards

A data dependency will occur when an instruction tries to read a register in stage_decode which is being written by a previous instruction in stage_writeback. In upcoming checkpoint, we stall the pipeline using a detection hazard unit.

Between first and the next instruction in data hazard,

> if (id_ex.rd == if_id.rs1 || id_ex.rd == if_id.rs2) → 3 bubbles so that decode regfile has the updated value after WB of first instruction, ie, 3 cycles.

Between first and the next-to-next instruction,

> if (ex_mem.rd == if_id.rs1 || ex_mem.rd == if_id.rs2) → 2 bubbles

Similarly, for the next instruction

> if (mem_wb.rd == if_id.rs1 || mem_wb.rd == if_id.rs2) → 1 bubble

Data Forwarding

We will have to modify the current datapath to implement data forwarding. For example, the alu_out for the first instruction is available in alu_out_reg in the ex_mem barrier. This value is

already available, and we don't need to for the 1st instruction to complete its WB stage (or even stall). Therefore, a connection can be made between alu_out_reg of ex_mem and an ALU input.

Similarly, for other forwarding paths (MEM → EX, WB → EX, WB → MEM) we will require more connections and, that's why, we think we will have to implement another mux that takes multiple forwarding paths and the original rs1_out, rs2_out.