Bilkent University
Computer Engineering
CS 224

Design Report

Lab # 5

Section # 3

Ghulam Ahmed (22101001)

9 May 2023

Part 1. Preliminary Work

1b) The list of all hazards that can occur in this pipeline. For each hazard, give its type (data or control), its specific name (“compute-use” “load-use”, “branch” etc.), the pipeline stages that are affected.

Data Hazards

Register RW hazard: Decode and writeback stages are affected by this hazard.

Compute-use hazard: Execute, memory and writeback stages are affected by this hazard.

Load-use hazard: Execute, memory and writeback stages are affected by this hazard.

Load-store hazard: Execute and memory stages are affected by this hazard.

Control Hazards

Branch hazard: All stages are affected by this hazard.

1c) For each hazard, give the solution (forwarding, stalling, flushing, combination of these), and explanation of what, when, how

Data Hazards

This occurs when there is a data dependency between instructions. The solution is to freeze earlier pipeline stages until the data becomes available or to provide a bypass to get the data to the right stage.

Control Hazards

This occurs when the pipeline must stall because the next instruction to execute is unknown until the current instruction finishes executing. The solution is to speculate about the hazard resolution and kill the instruction later if the speculation is wrong.

1d) Give the logic equations for each signal output by the hazard unit, as a function of the input signals that come to the hazard unit. This hazard unit should handle all the data and control hazards that can occur in your pipeline (listed in b) so that your pipelined processor computes correctly

1. Forwarding logic for compute-use hazard:

   if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
   ForwardAE = 10
   else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
   ForwardAE = 01
   else
   FowardAE = 00

   For rtE replace rsE with rtE and Forward AE with Forward BE

2. Stall logic for Load-use hazard:

   lwstall = ((rsD == rtE) or (rsD == rsE) AND MemtoRegE)
   stallF = StallD = FlushE = lwstall

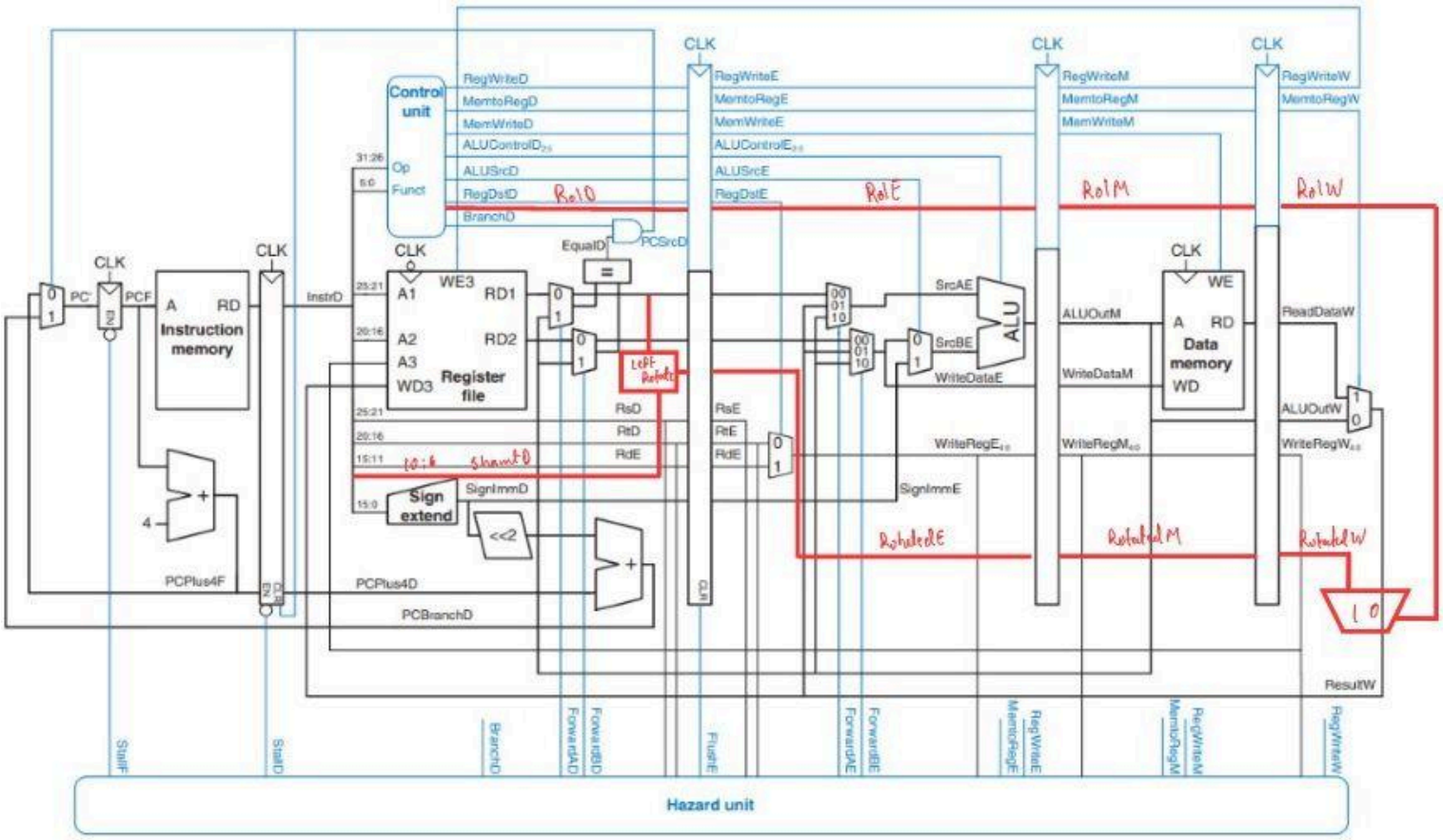
3. Forwarding logic for branch hazard:

   ForwardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM
   ForwardBD = (rtD != 0) AND (rtD == WriteRegM) AND RegWriteM

4. Stall logic for branch hazard:

   branchstall = (BranchD AND RegWriteE AND (WriteRegE == rsD OR WriteRegE == rtD)) OR (BranchD AND MemtoRegM AND (WriteRegM == rsD OR WriteRegM == rtD))
   StallF = StallD = FlushE = lwstall OR branchstall

Section 3 instruction: rol



1e) Write down all hazards the new instruction can cause, and their solutions.

Data Hazard: If the value in register $s1 (RS) is needed before the rotation operation is completed, this could result in a data hazard. This can occur if another instruction that depends on the value of $s1 is executed before the rotation operation is completed.

Solution: To avoid data hazards, the processor can implement techniques such as forwarding or stalling. With forwarding, the result of the rotation operation can be forwarded to the instruction that needs it, eliminating the need to wait for the rotation operation to complete. With stalling, the processor can delay the execution of subsequent instructions until the rotation operation is completed.

Control Hazard: If the rotation operation changes the value of the program counter (PC), this could result in a control hazard. This can occur if the rotated value is used as a branch target

address or jump address, and the rotation operation changes the address to which the program jumps.

Solution: To avoid control hazards, the processor can employ techniques such as branch prediction or branch delay slots. With branch prediction, the processor can predict the target address of a branch instruction before it is executed, allowing the next instructions to be fetched from the predicted target address. With branch delay slots, the processor can execute the instruction following a branch instruction before the branch is taken, allowing the next instruction to be executed immediately after the branch.