Bilkent University
Computer Science
Department
CS 224

“Design Report”
Lab 5
Section 6

Yasemin Akın
22101782

12 May 2023

b.

1. Compute-use (Read and Write, RAW) Hazard: Data Hazard, Execute and Writeback stages are affected.
2. Load-use Hazard: Data Hazard, Execute, and Writeback stages are concerned.
3. Load-Store Hazard: Data Hazard, Execute, and Writeback stages are affected.
4. Branch Hazard: Control Hazard, Fetch stage is affected.

c. The order of solutions given below is according to the order of the hazards written above.

1. This hazard occurs when an instruction writes a register and either of the 2 subsequent instructions reads that register. Below MIPS assembly code would cause this hazard if the precautions are not taken already:

   ```
   1       add    $t0, $t1, $t2
   2       sub    $t3, $t0, $t2
   3       and    $t4, $t0, $t2
   ```

   Stalling result-dependent instructions, which means that stalling instructions 2 and 3, or Forwarding needed results from proper positions, which means that forwarding the new $t0 value from the start of DM Stage to the second instruction Execute Stage or from WB Stage to the third instruction Execute Stage, are the two leading solutions for this data hazard.

2. This hazard occurs when a lw instruction takes place, hazard is really similar to the previous hazard, which has been caused by reading the old value of a changed register. Below MIPS assembly code would cause this hazard if the precautions are not taken already:

   ```
   1       lw     $t0, 48($sp)
   2       add    $t1, $t0, $t2
   ```

   We should use Stalling by keeping the second instruction on hold till the first instruction completes the M Stage and then use Forwarding to forward the resulting data of the first instruction’s M Stage to the second instruction’s Execute Stage.

3. This hazard is very similar to the second hazard. The difference is that it happens when a sw instruction follows a lw instruction. Below MIPS assembly code would cause this hazard if the precautions are not taken already:

   ```
   1       lw     $t0, 48($sp)
   2       sw     $t0, 0($t1)
   ```

The hazard is that the $t1 register will not have the new value of the $t0 register. As a solution, Forwarding can be used to forward data (a feedback mechanism) from WB Stage to M Stage.

4. We may use Stalling and stall the subsequent instructions for 3 clock cycles (until the end of the beq instruction), but stalling for 3 clock cycles is degrading. Therefore, we should use Flushing, which means that we continue fetching subsequent instructions like the branch is not taken, and at the end of the beq instruction, if the branch was supposed to be taken, we flush the unnecessary instructions between. However, too much Flushing is also degrading, so we should probably combine Forwarding and Flushing and decide if the branch is going to be taken as early as it can be (for instance in the Decode Stage) and then forward the result to the Fetch Stage then flush the reduced unnecessary instructions between.

d.

1. Forwarding Logic for ForwardAE:
   ```
   if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
   ForwardAE = 10
   else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
   ForwardAE = 01
   else ForwardAE = 00
   ```

2. Forwarding Logic for ForwardBE:
   ```
   if ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM) then
   ForwardBE = 10
   else if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW) then
   ForwardBE = 01
   else ForwardBE = 00
   ```

3. Load Stalling Logic:
   ```
   lwstall = ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE
   StallF = StallD = FlushE = lwstall
   ```
   Load Forwarding Logic uses 1 and 2 above.

4. Forwarding Logic for ForwardAD:
   ```
   ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
   ```

5. Forwarding Logic for ForwardBD:
   ```
   ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
   ```

6. Stalling Logic for Branch:
   ```
   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)

e.

1. Data Hazard - Read After Write (RAW) Hazard (first hazard in option a): We should use either the Forwarding or the Stalling technique. Below MIPS assembly code snippet can cause RAW hazard:

   ```
   1       label:
   2               la      $s0, label
   3               addi    $s1, $s0, 4
   4               add     $t0, $s0, $s1
   5               jr      $t0
   ```

   The RAW hazard here can be solved by Forwarding the result of the ALU operation from the Memory Stage of the third instruction to the Execution Stage of the forth instruction.

   If we use jr instruction after a lw or sw instruction, there will also be a RAW hazard but in this case, we need to solve this hazard by a different technique since these instructions have 2-cycle-latencies. Below MIPS assembly code snippet can also cause a RAW hazard:

   ```
   1       label:
   2               la      $s0, label
   3               lw      $t0, 0($s0)
   4               jr      $t0
   ```

   Here, we should also use Stalling besides Forwarding because we can not forward backward. We should stall jr instruction (which is the second instruction above) for 1 clock cycle and then forward the data at WB Stage of the first instruction to the E Stage of the second instruction as SrcA.

   Below MIPS assembly code snippet can not cause any hazards:

   ```
   label:
           addi    $s1, $zero, 20
           addi    $s2, $zero, 34
           add     $t0, $s1, $s2
           la      $t0, label
           sub     $t1, $s1, $s2
           or      $t2, $s1, $s2
           jr      $t0
   ```