Bilkent University Computer Engineering Department

Computer Organization(CS224) LAB5 Design Report

Name: Utku Kurtulmuş

ID: 21903025

Section: 2

LAB date: 13.04.2022

B)

DATA HAZARDS

There are two main data hazard that can occur in this pipeline:

RAW Hazard: This hazards occurs when we try to read a value from register file that is not written yet to the reg file. Decode stage is affected.

Load-Use Hazard: This hazard is similar to the RAW hazard but it's special for load operations. Load operations takes more time and we will need to handle load-use hazards specially. Execute and memory stage is affected.

CONTROL HAZARDS

Branch Control Hazard: This hazard occurs when we use branch instructions. Since we haven't decide which stage to go when we fetch the next instruction, we might execute instructions that shouldn't be executed. In normal cases branch decision made in memory stage so the stages before that are affected by this but in this complete design we use early branch prediction and handle most of the problem.

C)

Solutions

RAW HAZARD: Forwading solves the problem. In execute state, we will need multiplexers to choose if we are forwarding data. We might forward data from the alu or data memory(ResultW). We decide when to forward using Hazard Unit, If the conditions are causing RAW hazard we forward data.

LOAD-USE HAZARD: This hazard is similar to the Raw hazard, but Forwading won't be enough to solve this problem. We can't forward data from data memory to ALU if the next instruction is coming just after lw instruction using the register that we will load to. If there is a one instruction between them there is no problem, we can use forwarding and solve the data. So we will use stalling as well to solve this issue. The next instruction needs to wait 1 clk cycle.

BRANCH CONTROL HAZARD: We solve this hazard with early branch prediction. It is also implemented in this design. Normally we decide the branch control signal in memory stage and we had to take 3 instruction to pipeline till we came to memory stage, and if we miss predict the branch instruction we flush this 3 instruction.(clear registers). Instead of deciding it in the memory stage we can make eary branch decision in the decode stage so we can flush just one instruction if we mispredict the result.

D) Hazard Logics

```
if ((rsE != 0) & (rsE == WriteRegM) & RegWriteM) begin //RAW hazard
   ForwardAE = 2'b10;
end
else if ((rsE != 0) & (rsE == WriteRegW) & RegWriteW) begin
   ForwardAE = 2'b01;
end
else begin
   ForwardAE = 0;
end

if ((rtE != 0) & (rsE == WriteRegM) & RegWriteM) begin //RAW hazard
   ForwardBE = 2'b10;
end
else if ((rtE != 0) & (rsE == WriteRegW) & RegWriteW) begin
   ForwardBE = 2'b01;
end
else begin
   ForwardBE = 0;
end

//For Load use hazards, Stalling needed
lwstall = ((rsD == rtE) | (rtD == rtE)) & MemtoRegE;

//solving control hazards(We need both forwading and stalling)
ForwardAD = (rsD != 0) & ( rsD == WriteRegM) & RegWriteM;
ForwardBD = (rtD != 0) & (rtD == WriteRegM) & RegWriteM;

branchstall = (BranchD & RegWriteE & (WriteRegE == rsD | WriteRegE == rtD) )
                                                |
           (BranchD & MemtoRegM & (WriteRegM == rsD | WriteRegM == rtD) );

FlushE = Stallf = StallD =  lwstall | branchstall;
```

E)

We will use a similar design as we did in lab 4 for the srrac operations. We will need a new port in the register file for the RD register and we will read the data from that port. However, on execute stage we need to be careful, If the data is not written yet back to the register file and we read the wrong value RAW hazard occurs, also load-use hazard can occur. Hence, we will implement the same design as we did for RS and RD for the srrac operation and put one more 4 to 1 mux, and use forwading and if necessary stalling.

```
NO HAZARD TEST
addi $a0, $zero, 5
addi $a1, $zero, 12
addi $a2, $zero, 2
sracc $a0, $a1, $a2  #$a0 = 12 / 4 + 5 = 8
Hazard Test  1
addi $a1, $zero, 12
addi $a2, $zero, 2
addi $a0, $zero, 5  //Value of rd register not settled yet
sracc $a0, $a1, $a2  #$a0 = 12 / 4 + 5 = 8
Hazard Test 2
addi $a3, $zero,5
sw $a3, 0($a7)
addi $a1, $zero, 12
lw $a0, 0($a7)  // DM[$a7] = 5
addi $a2, $zero, 2
sracc $a0, $a1, $a2  #$a0 = 12 / 4 + 5 = 8
```