Bilkent University

CS Department

CS 224 Computer Organization

22003250

Design Report

Lab 5

Section 1

İpek Öztaş

22003250

15 May 2023

Question 1: 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”, “load-store”, “branch” etc.), the pipeline stages that are affected.

- Compute-use Hazard (Data Hazard)

Pipeline stages affected: Decode stage of the instruction depending on the previous instruction’s result will fetch wrong data value. Therefore, Execute and WriteBack stages will perform operations by using wrong data value and will be affected.

- Load-use Hazard (Data Hazard)

Pipeline stages affected: Possibly Execute and Memory (in case memory write) stages can be affected by this two cycle latency.

- Load-store Hazard (Data Hazard)

Pipeline stages affected: Memory stage of storing instruction. This happens because wrong data will stored in memory location.

- Branch Hazard (Control Hazard)

Pipeline stages affected: Since there is a delay at branch prediction there will be 3 unnecessary instructions that are fetched in the case of branch misprediction. Fetch, Execute, Memory, WriteBack and Decode stages are affected.

Question 2: For each hazard, give the solution (forwarding, stalling, flushing, combination of these), and explanation of what, when, how.

- Compute-use Hazard (Data Hazard)

How this hazard occurs?

An instruction depends on the result of a prior instruction that is still in the pipeline. For instance RAW hazard is an example for this type since the data is read after write wrong data can be read. The main reason is after the execute stage data computed is not written until the end of the write back stage.

In the Decode stage of a subsequent instruction wrong data from the previous instruction’s destination register can be read since the correct data is not written in the source register of the next instruction yet.

When this hazard occurs?

An example for this hazard is the R-type add instruction. R-type instruction’s destination register (rd) not yet written when following instruction’s source registers (rs) or (rt) request to access (rd) of previous instruction which is not written yet.

add r1, r2, r3

sub r4, r1, r3

in this example add instruction writes back its data during the WriteBack stage but the sub instruction reads its data in Decode stage which happens beforehand. Therefore, sub instruction may read an old (wrong) value of r1.

Solution for this hazard:

Solution for this hazard is called Forwarding/ bypassing which means it is not needed to wait for the data to be written back to the register file before using it but instead it is used immediately by forwarding the result from one pipeline stage to the next. Therefore, we don't need to wait for the WriteBack stage. Data can Forwarded to the next instruction's Execute stage for enabling the following instruction use correct data value. Another solution might be Stalling.

• Load-use Hazard (Data Hazard)

How this hazard occurs?

Instructions that require memory reading cannot read data values until the Memory stage is completed. As a result, subsequent instructions will be unable to access data loaded from memory by prior instructions at the Execute stage.

When this hazard occurs?

For example in the following instructions

lw $t2, 20($t1)

and $t4, $t2, $t5

Since load word is completed after mem stage is finished at the same time and instruction needs the value of $t2 when it enters the execute stagei this hazard occurs.

Solution for this hazard:

Forwarding does not solve this problem. Stalling is a solution where pipeline hold until data is available.

• Load-store Hazard (Data Hazard)

How this hazard occurs?

When data wanted to store at a memory location just after it is loaded from memory.

When this hazard occurs?

When lw and sw instructions are used consecutively with the same rt register.

For example in the following instructions

lw $t2, 20 ($t1)

sw $t2, 10($t3)

Since the second instruction wants to store the data which is loaded in the previous instruction to a memory location, this hazard occurs.

Solution for this hazard:

Stalling is a useful technique for deferring memory loading until the next instruction's Decode step, when it will fetch data from the same register.

- Branch Hazard (Control Hazard)

How this hazard occurs?

This hazard is caused because branch decision is not made by the time next instruction fetched from the instruction memory. The branch prediction will be chosen at Memory stage and this causes delay.

When this hazard occurs?

When branch instruction is used.

Solution for this hazard:

Pipeline can be stalled for 3 cycles.

Additional hardware (equality comparators) can be used for branch decision at an earlier stage.

Flushing the fetched instructions is another solution to fix branch mispredictions.

Question 3: 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.

Data Forwarding

Logic of Hazard Unit for Forwarding

```
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
```

Logic of Hazard Unit for Stalling & Flushing

lwstall = ((rsD = = rtE) OR (rtD = = rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall

Control Forwarding and Stalling Logic

Forwarding Logic

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

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

Stalling Logic

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)

Question 5: Think about extending the pipelined processor with the instruction corresponding to your section on the table given below. The instructions are either identical or very similar to the ones you implemented in Lab 4. Write down all hazards the new instruction can cause, and their solutions. You can refer to the hazards you listed in Part 1-b if some of them also occur with the new instruction. Then, for each hazard you listed, write a small test program in MIPS assembly that will show whether the pipelined processor is working even in the presence of the hazard. You should also write a test program with no hazards. The tests should verify that the new instruction works correctly under any circumstances. Note that, your design should avoid stalls if it is possible to use forwarding. Any kind of extra stall unnecessarily will cause you to lose points.

In the lui instruction implementation, data hazard can occur when the result of a lui instruction is used in a following ALU operation or an operation that requires memory access. The second instruction tries to access the register value that has not been completed yet.

The hazard occurs due to the decode stage. This also effects the following stages: decode, execute, memory, writeback.

Forwarding can be used to avoid stalls. It provides the result of the lui instruction directly to the following instruction.

- Compute-use hazard (ALU): occurs when a following ALU instruction uses the result of lui.
  Example test program:
  lui $t1, 0x1001
  addi $t0, $t1, 5
- Compute-use hazard (Memory access): occurs when the following is a lw/sw instruction since they both require memory access.
  Example test program:

  lui $t0, 0x1001

  sw $t1, 0($t0)

Test program without hazards:

```
lui $t0, 0x1001

nop

addi $t1, $t0, 5
```