Rawisara Chairat

U65053623

Lab Report

Introduction

The goal of this lab was to understand and implement the functionalities of a pipelined MIPS CPU. Starting with a five-stage pipeline without forwarding, additional features were added, including "1 ahead" and "2 ahead" forwarding, arbitration logic, and hazard control. Testing each feature helped ensure accuracy and reliability in the CPU's handling of data dependencies.

Implementation

1. **Synthesize the Project and Generate Outputs**

   The initial code was synthesized to observe the base behavior of a standard five-stage MIPS pipeline with no forwarding.

2. **Add Forwarding Unit**

   Implemented forwarding unit to detect and address data hazards for instructions that require data from the immediately preceding instruction. This was done using the ForwardA and ForwardB signals to direct values from MEM and WB stages to the ALU inputs.

3. **Add "1 Ahead" Forwarding**

   In this section, we implemented a 1-Ahead forwarding mechanism to resolve data hazards that occur when the destination register in the EX/MEM stage matches either source register in the ID/EX stage. This allows the pipeline to use the most recent data without needing to stall. Forwarding is triggered if (EX/MEM).Rd matches either (ID/EX).Rs or (ID/EX).Rt.

4. **Add "2 Ahead" Forwarding**

   This section covers the implementation of the 2-Ahead forwarding mechanism, which resolves data hazards by forwarding values from the MEM/WB stage to the ID/EX stage. This allows recently written data to be used without waiting for a full write-back. Forwarding occurs if (MEM/WB).Rd matches either (ID/EX).Rs or (ID/EX).Rt.

5. **Add Arbitration Logic for Decoding Between 1 & 2 Ahead**

   In this section, we implemented arbitration logic to decide between 1-Ahead and 2-Ahead forwarding, prioritizing the most recent data available. This ensures that the pipeline receives the correct values when multiple forwarding paths are possible.

   4.1. **Initialization:**

   - ForwardA and ForwardB are initially set to 00 to indicate no forwarding.

   4.2. **1-Ahead Forwarding (EX/MEM)**:

   - If (EX/MEM).Rd matches (ID/EX).Rs, set ForwardA = 10, as this indicates the data needed by Rs is in the EX/MEM stage.
   - Similarly, if (EX/MEM).Rd matches (ID/EX).Rt, set ForwardB = 10 to forward the data to Rt.

   4.3. **2-Ahead Forwarding (MEM/WB)**:

   - If (EX/MEM).RegisterRd does not match (ID/EX).Rs, but (MEM/WB).Rd matches (ID/EX).Rs, set ForwardA = 01. This provides the data from MEM/WB to Rs if it isn't available from EX/MEM.
   - Likewise, if (EX/MEM).RegisterRd does not match (ID/EX).Rt, but (MEM/WB).Rd matches (ID/EX).Rt, set ForwardB = 01.

Rawisara Chairat
U65053623

6. **Add Logic for $0 Write**

The $0 Write logic ensures that register $0 is handled correctly within the pipeline. Since register $0 is hardwired to zero in the MIPS architecture, it must always contain a value of 0, regardless of operations in the pipeline. To maintain this constraint, specific logic is implemented to avoid forwarding or writing to $0. If the destination register (Rd) is $0, no forwarding should occur. This is checked by evaluating if (Rd == 0), where Rd represents the destination register in either the EX/MEM or MEM/WB stage.

7. **Add Logic for No Write**

The No Write logic manages situations where no forwarding is needed because a register write is not intended. By checking the RegWrite signals in the EX/MEM and MEM/WB stages, the forwarding unit can avoid unnecessary data forwarding when no write operation is active.

- **No 1-Ahead Forwarding:** If (EX/MEM).RegWrite == 0, then no 1-Ahead forwarding is performed from the EX/MEM stage.
- **No 2-Ahead Forwarding:** If (MEM/WB).RegWrite == 0, then no 2-Ahead forwarding is performed from the MEM/WB stage.

8. **Check Register Bypass Works**

The Register Bypass logic directly forwards recently written data to the registers that require it, without waiting for it to propagate through the pipeline. This bypass logic checks if the registers being read (ReadReg1 and ReadReg2) match the most recent write register (WriteRegister) and, if so, bypasses the value from WriteData directly to ReadData1 or ReadData2. We verified that the bypass mechanism worked as expected across various instruction sequences. Ensured data hazards were avoided by implementing bypass control signals effectively.

Results

Here is how the datapath should look like after the forwarding unit is being added to the initial datapath.

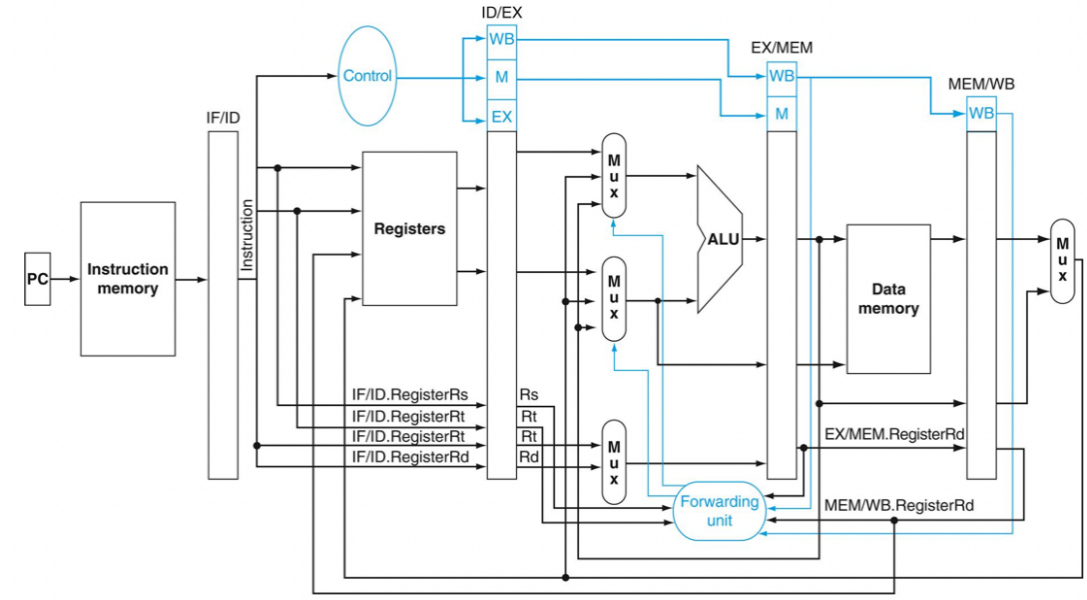


Diagram 1: The datapath modified to resolve hazards via forwarding.

Rawisara Chairat

U65053623

Testing

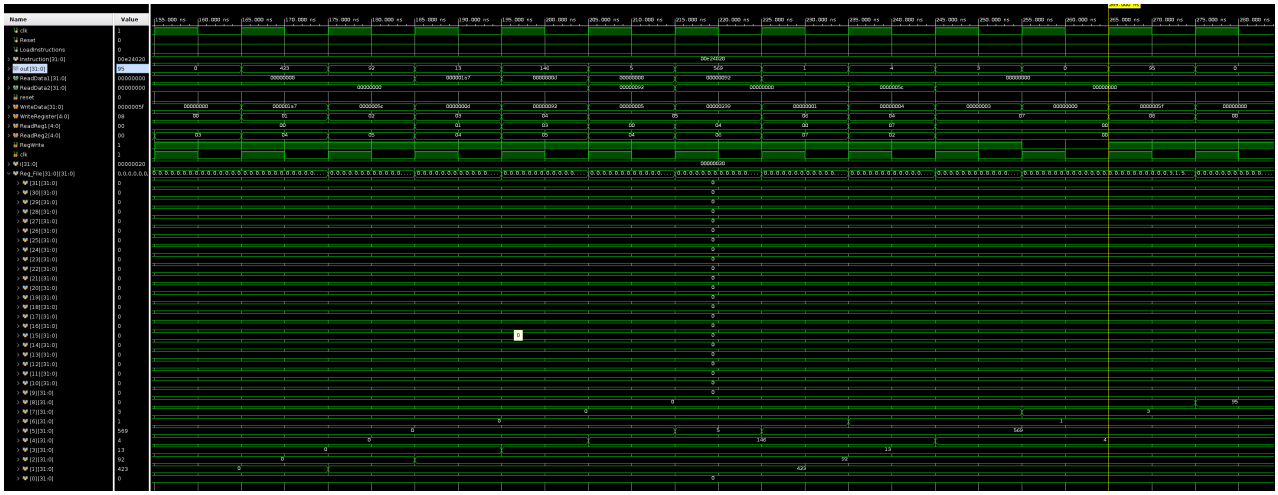
Initial Waveform Analysis (Without Forwarding)

After generating the initial waveform without implementing forwarding, it was observed that the pipeline produced incorrect results in certain cycles. This error occurred because the pipeline did not account for data hazards—situations where subsequent instructions depended on the results of previous instructions that had not yet completed the write-back stage. As a result, values read from the register file were stale, leading to inconsistencies in the output.

Waveform 1: Initial Pipeline Waveform

Modified Pipeline with Forwarding Enabled

To address this issue, I modified the pipeline by adding forwarding logic. This addition allows the pipeline to bypass data directly from later stages (such as EX/MEM and MEM/WB) to earlier stages (ID/EX), ensuring that dependent instructions receive the most recent data without waiting for a full write-back. After regenerating the waveform with this enhancement, the corrected results were clearly observed. The modified waveform confirmed that the forwarding mechanism effectively resolved the data hazards, resulting in accurate values being computed and passed between pipeline stages.



Waveform 2: Modified Pipeline Waveform including the hazard control

Conclusion

The goal of this lab was successfully achieved by implementing a forwarding unit to resolve data hazards in a five-stage MIPS pipeline. Initially, without forwarding, the pipeline exhibited incorrect results and performance degradation due to the need for stalls. By adding 1-Ahead and 2-Ahead forwarding mechanisms, we were able to eliminate these stalls and ensure that data required by later instructions was available at the right time.

Citation

Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: The hardware/software interface (MIPS Edition)* (5th ed.). Morgan Kaufmann. Figure 4.56.