CPE380 Project 4: Pipelined MIPS Processor

Implementor's Notes

Michael Naughton and Elijah Gray
University of Kentucky, Lexington, KY USA
mcna226@uky.edu
elijah.gray@uky.edu

11/24/2025

ABSTRACT

This project was focused around modifying an existing implementation of a pipelined MIPS processor. The goal was to gain experience with designing pipelined processors by implementing several instructions. We added instructions for the NOR variable shift (sllv,srlv,srav) as well as BNE. We also had to extend the decoder and add ALU functionality.

1 General Approach

The first instruction to implement was the MIPS **nor**, following the format **nor $rd, $rs, $rt**. This required defining the a new funct field for **nor**, a new ALU code, a new decode statement for the ID (instruction decode) stage, adding the ALU code to EX (execute) stage, and finally adding a **$display** statement for **nor** to the state-by-state trace. All of these steps were necessary for all of the new instructions. In the EX stage, the output of the ALU was set to equal $\sim (ID_s | ID_{src})$.

The **srlv** and **srav** instructions were similar to the **nor** instruction. The only difference being how they were implemented in the ALU during the EX stage. For the **srlv** instruction it was as simple as using the built in "»" operator. For **srav** a conditional had to be introduced. If the most significant bit was a '0', then the shift right operator sufficed. If, however, the MSB was a '1', then the number being shifted had to be sign extended to 64-bits before being shifted.

The **sllv** instruction required shifting left using the value of the **rt** register, masked to the lower 5 bits, i.e. $ID_s << ID_t[4:0]$. Forwarding and stalling logic had to be accounted for to handle dependencies when the source registers were being written to in previous cycles.

For the **bne** instruction, branch comparison was performed in the ID stage by checking if $s \neq t$. If the branch was taken, the instruction in IF was squashed, and the PC was updated to the branch target. Forwarding also ensured the correct register values were used during branch resolution to prevent hazards.

2 Issues

For the **srav** instruction, the initial solution used involved putting a **reg** declaration within an **always** block. This may have worked in SystemVerilog, but is invalid in Verilog.

3 Testing

For readability, a new statement was added to the beginning of each trace that prints the value in register 4. Most of the tested instructions output to register 4 so that the result of that instruction can be read more easily. The added test cases can be found in **m[6:0]**. First the **nor** instruction is tested with the values **601** and **11811** which should result in **r[4] = 4294955396** by the time the PC reaches **20**.

Next, **sllv** is tested, shifting **42** left by **1** which should result in **r[4] = 84** immediately after the result from the previous instruction.

Next, **srlv** is tested, shifting **88** right by **1** resulting in **r[4] = 44**.

Next, **srav** is tested, first by shifting **11811** right by **1** resulting in **5905**. Then, **-11769** (or **4294955527** unsigned) is shifted right by **1** resulting in **-5885** or (**4294961411** unsigned).

Finally, **bne** is tested by comparing the value stored in **r[6]** with the value in **22**. Initially **r[6] = 88**. If **r[6] != 22** the program branches back one instruction. The previous instruction halves **r[6]** and then compares it to /textbf22 again. Once **r[6] == 22**, the program continues.