Assignment 3 Report

Nicholas Petosa

Design Choices

This pipeline has 3 stages, a combined IF/ID/RR, followed by EX, then a combined MEM/WB. In general, the control signals passed down the pipeline are as follows: isnop (whether this stage contains a NOP bubble), aluimm (whether to put sxtimm in aluin2), alufunc (function to use at the ALU), wrmem (whether this is a store), ldmem (whether this is a load), wrreg (whether we are writing to a register), destreg (if we are writing, which register to write to).

The PC is controlled by a MUX, which either sets it to STARTPC on reset or sets it to an incremented PC if the pipeline is not stalled. For bubbles, we have an isnop flag for each stage of the pipeline. We figured this would be easier to implement than clearing all registers for a NOP bubble, since we would only need to set and check 1 bit as opposed to flushing all bits in all buffer registers. For the ALU, we added a JAL instructions, we just append 2 zero bits to the front of their OP1 code to access correct performance to the what the frame provided, which just sets aluout to the incremented PC. There is also a control signal to determine whether to load RT or sxtimm into aluin2. For BR and ALUI For stalling logic, we stall the pipeline when EX and MEM/WB are not NOP and if the instruction in IF/ID/RR has an RT or RS register which matches the destination register of the instruction in EX or MEM/WB. We use static branch prediction and flush the first stage (set it to NOP) when we are incorrect.

Problems/Issues

To aid in debugging, we implemented several techniques. Firstly, modified the clock pulse so that we could either have it tick once per second or every time we pressed KEY. The former was better for examining the progress of large programs like test.mif while the latter was more helpful in debugging specific instructions. Since we had the assembler already built from assignment 2, we were able to generate small programs to debug certain instructions. We used the HEX display to give use a visualization of the current state of the pipeline. We had it display what instructions were in each stage of the pipeline at any time. Additionally, we used the LEDR display to indicate which stages contained a NOP and whether the pipeline was stalled.

In terms of issues, one big one we faced was a combinational loop. This is when the output of a wire is, by some direct or indirect path, fed back into its input. This leads to oscillation and unstable/unreliable behavior. Quartus offered some insight into the issue, indicating the combinational loop involved 6 nodes. By removing one of those nodes, the error went away and performance stabilized. The node we had to remove was a control signal that specified whether the current instruction required only RS for stalling. As a results, our pipeline may insert one more NOP than is actually necessary for JAL, LW, and ALUI instructions. We decided to leave this because when we implement data forward in assignment 4, this detection will be largely unimportant; data will just be forwarded to an unused register.

Contribution: 50%

I implemented the pipeline structure as well as ALUR, ALUI, SW, and LW instructions, as well as MMIO.

Diagram: See attached files: pipeline.jpg, stall-controller. jpg, IFIDRR. jpg, and AGEX+MEMWB. jpg.