Tara Sothy and Matt Westerhaus

Team 5

Simulator Behavior

Our simulator is able to read in instructions converted from assembly language into machine code. The simulator is also designed to pipeline the instruction increasing our performance significantly. This means we start an instruction, and we aren't waiting for it to finish before starting the next instruction. We run instructions in parallel when resources become available to use. This allows us to overlap instructions to decrease the overall execution time. Each instruction is broken into five stages where each stage is completed in one cycle. Each cycle the simulator reads in an instruction and executes one stage of each instruction in flight. After each cycle the simulator prints out the state of the pipeline and once the program it is completed, it prints out the final state with the number of cycles, instructions fetched and retired, branches, and mispredictions.

Struggles and Shortcomings Reflection

- Our fetched and retired variables were wrong, we realized this was because we were misunderstanding when fetched is changed; when an instruction is fetched, we increment fetched but when instructions are overwritten with a noop we should be decrementing fetched so it doesn't count the same instruction twice and we should be decrementing retired, so it doesn't count the noops. For retired, we realized that the halt is executed in the second to last stage which is a problem because it is incremented in our last stage so to be able to count the halt instruction, we had to account for this when we initialized retired.

- When we initially made the two statetype structs, we allocated the same memory to both of them instead of allocating them each with their own memory.
- We also struggled when incrementing pc but quickly realized we were incrementing pc twice in one cycle.
- Our ADD and NAND instructions weren't behaving as expected this was due to a lot of smaller problems however the biggest was some variables weren't always being calculated, as an example aluresult.
- When implementing data hazards, we were unsure if we were supposed to use a temporary variable to hold the value to be overwritten or in each instance, if we were supposed to store the destReg value in a temporary variable which later might be stored in regA and/or regB variables within the IDEX struct. The latter is true.
- We also encountered a problem with load stalls, this was because we had load stalls "executed" at the beginning of the execution stage rather than at the end. This was a problem because if we had load stalls at the beginning it would execute the instructions in the execution stage twice.