Shawn Zhai 1006979389
Victor Wu 1007069039

Total numbers of cycles with Tomasulo

EIO trace	sim_num_tom_cycles
gcc	1681443
go	1695064
compress	1851550

Each stage code description

1. **Fetch and Dispatch Stage (**fetch **and** fetch_To_dispatch**):**
   The fetch function reads instructions from the trace and places them in the instruction queue (IFQ) if space is available, ensuring in-order fetch. Instructions are fetched in program order. We mark the start of the dispatch stage of an instruction the cycle it enters IFQ.
   Special cases: Trap instructions (IS_TRAP) are skipped.

2. **Issue Stage (**dispatch_To_issue**):**
   The issue stage places instructions from the IFQ into the appropriate reservation stations based on type INT/FP and availability. Dependencies are updated here, linking each instruction to its producing instructions in the map table. If an instruction is successfully issued, the head of the IFQ is popped. We mark the start of the Issue stage of an instruction the cycle it enters the reservation station. The start of the Issue stage and Dispatch stage is ensured to be different.
   Special cases: Conditional and unconditional branches are removed from IFQ without entering reservation stations. Store instructions skip map table updates as they do not write to the register file.

3. **Execute Stage (**issue_To_execute**):**
   Instructions in reservation stations check for resolved dependencies and available functional units. If conditions are met, the instruction is sent to a functional unit and begins execution in the next cycle. Instructions are issued by age (earliest dispatch goes execute first in FU contention) and multiple instructions can start execution in the same cycle. In the reservation station, only instructions that have not yet started execution are considered. We mark the start of the Execute stage of an instruction the cycle it occupies an FU. The start of the Execute stage and Issue stage is ensured to be different.

4. **Execute to Writeback Transition (**execute_To_CDB**):**
   Instructions complete execution in their functional units based on latencies provided. Completed instructions attempt to move to the CDB. Instructions are sent to CDB by age (earliest dispatch uses CDB first in contention). We clear the RS and FU entry of the instruction that goes to CDB. Only one instruction can

use CDB in a cycle and instructions wait in the Execute stage if CDB is not available. We mark the start of the CDB stage of an instruction the cycle after it finishes the Execute stage.
Special cases: Store instructions do not use the CDB, and are directly removed from functional units and reservation stations post-execution with the helper function free_store().

5. **Writeback Stage (**CDB_To_retire**):**
   Instructions on the CDB broadcast results to clear dependencies in reservation stations and update the map table.

Correctness of your code

We printed Tomasulo cycle values of different instruction counts (50, 100, 150) and different benchmarks. If Tomasulo works as intended, we should observe in-order dispatch, in-order issue, OoO execution, and OoO writebacks. This is exactly what we saw. We also manually traced the first 50 lines of the Tomasulo table for all three EIO traces, and observed that all data hazards and structural hazards were respected. We identified the special cases highlighted in the pdf (branching, jumps, stores) and their effects are observed in the printed table (branching/jumps only have dispatch cycle value, stores do not have CDB broadcast cycle value). With these observations, we are confident that our implementation is correct.

Toughest bug to solve

1. **Repeated back & forth with an off-by-1 problem + order of Tomasulo operations**
   Originally our code retired the CDB first, before calling issue_to_execute. By doing this, we account for the fact that an instruction waiting for a dependency in CDB can finish the Issue stage in the same cycle as the CDB retires.

```
CDB_To_retire(cycle);
fetch_To_dispatch(trace, cycle);
dispatch_To_issue(cycle);
issue_To_execute(cycle);
execute_To_CDB(cycle);
```

   However, within issue_to_execute, if an instruction gets its dependency from the CDB and finishes the issue stage in a cycle, it has to start the Execute stage in the next cycle. To resolve this, we made the instruction's tom_execute_cycle = current_cycle + 1. But doing this is problematic, we identified with manual tracking that there are instructions in the printed Tomasulo table where the execution cycle is 1 more than what it is supposed to be. And we found out that this is because there are instructions that do not wait for data hazards (register values), but structural hazards (FUs). If an instruction is waiting for FU to free up, it should be able to start the Execute stage (cycle num set in issue_to_execute) in the cycle when FU is available. Below is an situation where the above implementation causes error:

   instr1 waits for a free FU to execute in C10, instr2 frees an FU in execute_to_CDB in C10. FU is now re-allocatable in C11 and instr1 should start

the Execute stage in C11, but it actually starts executing in C12 due to the +1 offset for tom_execute_cycle.

To fix this, we removed the +1 offset and moved the call of CDB_To_retire to between issue_To_execute and execute_To_CDB. This resolves both instruction waiting in RS due to data hazards and structural hazards.

2. **Incorrect understandings of what the functions are suppose to achieve**
   It took us a lot of time to wrap our heads around what exactly is required in each function to simulate Tomasulo for this lab. In lecture, we learned Tomasulo in a way that we track the completion cycle for the stages for each instruction, but we need to track the start cycle in this lab. For a long time we were confused by this discrepancy, trying to track the cycle numbers of stage completion. This resulted in incorrect implementations due to doing things in the wrong order and wrong functions, especially for between dispatch_to_issue and issue_to_execute. For example, in the early stage of coding, We kept instructions in dispatch_to_issue until it finishes the Issue stage and kept instructions in issue_to_execute until it finishes the Execute stage. In both functions, we had conditional checks for RS and FU availability, which did not make any sense and caused a lot of seg faults and errors in cycle numbers.

Work done by each partner

Shawn Zhai: coding, debugging, manually tracing Tomasulo table to ensure correctness
Victor Wu: coding, debugging, writing report