ECE 552 Lab 3 Report

Zhi Heng (Tony) Zhou & Jin Tao (Austin) Xu

Results from running:

sim-safe -max:inst 1000000 /cad2/ece552f/benchmarks/gcc.eio
sim-safe -max:inst 1000000 /cad2/ece552f/benchmarks/go.eio
sim-safe -max:inst 1000000 /cad2/ece552f/benchmarks/compress.eio

	# Cycles to Complete
gcc.eio	1681443 cycles
go.eio	1695064 cycles
compress.eio	1851550 cycles

Code description:

fetch_To_dispatch (Fetch)

Algorithm: This stage adds one instruction per cycle from the trace into the Instruction Fetch Queue (IFQ). It first checks if the queue is full or if all instructions have been fetched. If either is true, it stalls. Otherwise, it fetches the next instruction and, if valid, enqueues it.

Special Handling: If the fetched instruction is a trap, it is discarded, and the stage immediately tries to fetch the next instruction in the same cycle, ensuring traps never enter the pipeline. The first instruction in the trace is skipped as it is invalid.

dispatch_To_issue (Dispatch)

Algorithm: This stage moves the oldest instruction from the IFQ into a free Reservation Station (RS) and performs register renaming. It scans the appropriate RS bank (Integer or FP) for an empty slot. If a slot is free, the instruction is dequeued from the IFQ and placed in the RS. The register renaming logic then runs checks the map table for its source operands, marking them as "ready" or "pending" (by linking to the instruction that will produce them). Finally, if the instructions write a result, it updates the map table to show that it is now the producer for that destination register.

Special Handling: Branches do not use an RS; they are simply dequeued from the IFQ and effectively retired. Memory instructions are treated as integer operations and are placed in an integer RS.

issue_To_execute (Issue)

Algorithm: This stage iterates through each Functional Unit (FU). If an FU is free, it searches the corresponding RS bank for instructions that are ready and selects the oldest instruction. The ready check is done via a helper function called CanExecute which checks if all its dependencies are resolved and is not already executing.

execute_To_CDB (Execute)

Algorithm: This stage finds instructions that have completed their execution and arbitrates for the Common Data Bus (CDB). It scans all FUs, checking if an instruction's execution latency has passed. It then selects the single oldest (by program order) completed instructions that need to write a result. This instruction's RS and FU slots are freed, and it is placed onto the CDB.

Special Handling: Instructions that do not write results (like stores) are handled separately: once their execution is complete, their RS and FU slots are simply freed, and they do not compete for the CDB.

CDB_To_retire (Writeback)

Algorithm: This stage broadcasts the result from the instruction on the CDB to all waiting components. If the CDB is not empty, the function iterates through all RSs and the Map table, checking if any pending entries are waiting for the instruction on the bus; if so, it marks those operands as "ready."

Verification:

The verification is done using GDB. I put a break point on every time fetch_To_dispatch is called, and printed out the Map Table, Fetch Index, Reservation Stations, Function unit and Common Data Bus. I wrote them down and manually deduced what should happen in the next cycle and compared it to the actual result. I did this for 20 cycles to complete the verification.

Bugs Encountered:

The first bug I encountered was a problem with some dependencies not being cleared, this ended up being caused by the fact that I didn’t check if an instruction wrote to the CDB before populating the map table.

After getting the code running, the code still had a problem where the number of cycles to complete is widely higher than what was expected. The end cause being that there were no checks on the r_in and r_out register indexes. If a register is invalid, the index is set to -1, which accidentally adds a fake dependency between each line of the code.

Statement of Work:

Initial Code: Tony Zhou

Debug and Testing: Austin Xu

Report: Austin Xu & Tony Zhou