ECE552 – Lab Assignment 3: Dynamic Scheduling with Tomasulo

Andreea Varlan & Ben Pak

EIO TRACE	Total number of cycles with Tomasulo
gcc	**1670200**
go	**1686036**
compress	**1842838**

Note: I use a global counter 'done_insn_count' to keep track of the amount of insns that complete their process, which is incremented in different functions depending on the instr type

1) **Fetch_to_dispatch:** I used a circular buffer structure for IFQ which dispatches instructions in order of the last one arrived in the queue, and adds newly fetched instructions at the back of the queue in the first non-null position. I continue fetching until I encounter an insn that is NOT a trap (in case of a trap I incremented the done_insn_count since this is the last time we deal with this insn)
2) **Dispatch_to_issue:** I get the first insn at the top of the IFQ and check first if it is an unconditional/conditional branch. In case it is, I increment the done_insn_count and remove it from IFQ since branches do not proceed to Issue (complete D). If not a branch, I check if any reservation station entries are available, either of INT or FP type depending on the insn. If an RS entry is available, I check if the source operands are available (if their respective map table entries are free – which means they are ready to be read from the reg file). If source operands are not available, I write the Q entries on the instruction with the tag values of the results that will be received from different insns. I also write the output registers in the map table. I write the insn in the INT/FP reservation unit, remove it from the IFQ and set the issue cycle.
3) **Issue_to_execute:** At this stage, I wait for all RAW dependences to be resolved, if any, (by checking the Q entries of each reservation station entry that reached issue stage), and the availability of Functional Unit entries. The oldest instructions that are ready to begin execute can start executing. Since multiple instructions can start execute at the same time (max 3 INT insns as we have a total of 3 INT FUs, and only 1 FP), I tried to write a sorting algorithm to decide in which order multiple instructions could execute at the INT stage. I only managed to get two INT instructions to execute at the same time (and I will continue to work on this because I have become obsessed with this part of the function).
4) **Execute_to_CDB:** At this stage, I wait for instructions in the FUs to finish executing. As store instructions do not write to the CDB, once their execution finishes, the insn is ready to end its process, and thus we increment the done_insn_count and deallocate its RS and FU entries. For the rest of the instruction, we write the oldest of the insn(s) that finished execution to the CDB, and deallocate their RS and FU entries. We deallocate the RS and FU entries at this point because we want the instructions that have a structural hazard on RS or FU to be able to issue/execute in the same stage.
5) **CDB_to_retire:** Finally, in this stage we can write the CDB to all RS entries waiting to use that specific instruction by setting the Q values of the corresponding RS entries to

NULL, as well as the map table values at that specific register. At this point we can increment the done_insn_count since the CDB insn is ready to retire, and the CDB is reset.

- **is_simulation_done:** the simulation is completed when all instructions finish their respective process (either in retirement, end of execution, issue, dispatch or fetch)
- **HELPER FUNCTIONS:**
  - **is_RS_available:** used in dispatch_to_issue
  - **is_FU_available:** used in issue_to_execute
  - **remove_from_IFQ:** removes instruction from the head and shifts all remaining instructions to the front; used in dispatch_to_issue
  - **deallocate_RS_FU:** used in execute_to_CDB to remove entries from RS and FU

Correctness Testing

To test the correctness of my code, I first ran a small number of insns, as well as only the first 10 cycles, and observed the cycles populated in the Tomasulo Table. I spotted RAW dependences, structural hazards caused by lack of RS or FU entries and determined at which cycle each stage should occur. For example, I looked at the following two instructions:

```
addiu     r17,r29,4     4       5       6       11
addiu     r3,r17,4      5       6       12      17
```

Here I noticed the RAW dependency between two INT instructions, which means that the second addiu can start executing one cycle after the first addiu writes r17 to the CDB and retires. Also, I observed that the difference between the execute stage and writeback matches the latency for an INT op (5 cycles).

After comparing my Tomasulo table output for the first ~50 instructions with fellow classmates, I realized that after a certain point my execution stages would occur one cycle later than they should. This meant that I had to cover simultaneous execution of multiple instructions, which I managed to do for pairs of instructions only.

```
1 lw r16,0(r29)            1 2 3 8
2 lui r28,0x1003           2 3 4 9
3 addiu r28,r28,20912      3 4 10 15
4 addiu r17,r29,4          4 5 6 11
5 addiu r3,r17,4           5 6 12 17
6 sll r2,r16,2             6 8 9 14
```

In the 6th instruction above we can see that a structural dependency caused by the lack of an available reservation stations, which causes the issue stage to start on cycle 8 (instead of 7). This happens because issue for it can only start once an appropriate reservation station is freed, which occurs one cycle after the end of execution stage (7) of instruction 1.

Tough bugs

Some of my toughest bugs I experienced were very subtle. I realized I kept mixing up how I checked if an insn was NULL or not (I would use !insn instead of insn). This was a pesky bug because it seemed completely correct at first sight, and I had to write the whole statement out to convince myself of the correct condition.

Another hard bug to catch occurred when I would be stuck in an infinite loop. I wasn't able to get the Tomasulo table printed since the simulation would never finish, and with gdb I couldn't track the exact spot where this happens. Thus, I had to use detailed print statements throughout each of the stages to trace the complete process of each insn. This bug is something I am dealing with in my latest implementation in which I try to have 3 instructions able to execute in the same cycle if they have their source operands and functional units available.

Statement of work

Ben has implemented fetch_to_dispatch and is_simulation_done. Ben has also helped to debug at different stages. Andreea has covered everything else.