Lab 3 Report

Viet Minh Nguyen – 1007503989

1. **Cycles**

Total numbers of cycles with tomasulo:

- gcc.eio: **1681443**
- go.eio: **1695064**
- compress.eio: **1851550**

2. **Describe code:**

NOTE: When I refer to "matching tag", it means comparing pointers to the current instruction

- **CDB_to_retire:** if there is a value being broadcasted on commonDataBus, we will clear the tags of this instruction, if there's a match (RAW dependences) in other instructions' Q[0],[1], or [2]
    - Also need to clear the tag in any map_table entry that matches the CDB
    - Set CDB to NULL
- **execute_to_CDB**: This instruction iterates through both functional units to check if any instructions finished executing (based on given LATENCY and tom_execute_cycle when the instruction started executing), and add all these instructions to an additional **ready_instrs array**. From this, "Instructions that can proceed", as explained below, will be added to the **clear_rs_fu_instrs array**
    - Due to structural hazard, only 1 non-store instruction can be sent to the CDB. Therefore, only 1 such instruction can "proceed". However, since we want to prioritize the instruction that is older to be written to the CDB, the ready_instrs array is **sorted by instr->index.** Then, only the first non-store instruction can be allowed to proceed.
    - However, we need to also make sure that all store instructions that finished executing can also "proceed", since they do not need the CDB
    - An instruction that can "proceed" means an instruction that has its RS and FU entries cleared, which is done immediately at the end of this function. This is due to point 15, Lab 3 FAQ
- **issue_to_execute:** Iterate through the reservation stations, check which instructions are "ready", meaning all Q[0],[1],[2] is NULL. But similar to above, since we want to issue instructions by program order, I had to use a **ready_instrs array**, and implement **sorting** like above.
    - An edge case to consider is when an instruction has already been issued, in the next cycles that it has not finished executing, it's still in the RS. Therefore, the next iterations must avoid marking these instructions as "ready", by checking if tom_execute_cycle is 0.
- **dispatch_to_issue**:
    - Always try to dispatch the "oldest" instruction in the IFQ (since we want dispatch in order). Then, simply check for structural hazard in the corresponding RS.
    - For Unconditional and Conditional instructions, it is not sent to any RS like in the handout.
    - A hazard boolean is used. If there is a hazard, we do not de-allocate this instruction from the IFQ (which will lead to stalls later in the IFQ if IFQ is full). Else, **if instruction is not**

Unconditional/Conditional, we will copy tags from map table and assign tags to map table. Copy is done before changing map table, in case instructions like R1 + 4 -> R1 occur.

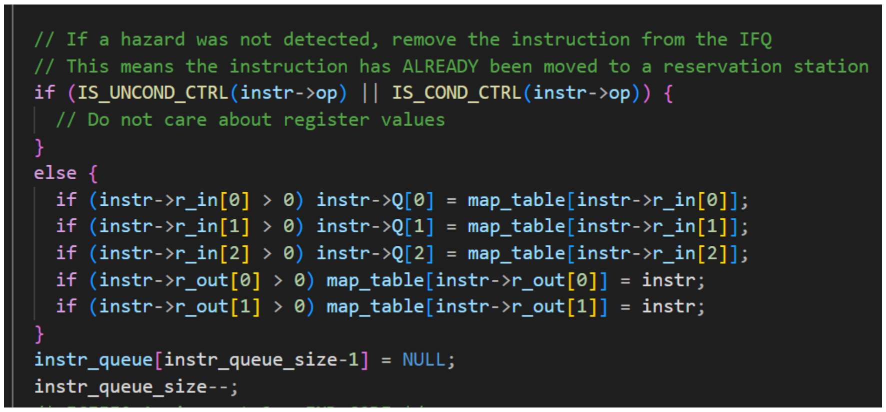
- **fetch_to_dispatch**:
  - To handle the case where the fetch_index ended, but pipeline is not empty (still executing), an if statement is needed to avoid additional fetching.
  - ONLY if the IFQ is not full, fetch() function is called. The function simply have a do-while loop to fetch until a non-trap instruction is fetched, then add it to the tail of the IFQ
- Finally, **is_simulation_done** simply iterates through and check whether the entire pipeline is empty. This denotes that the program has finished executing.

3. Tested correctness:

- First while testing, run the program with very small amount of instructions (30-50) then increases overtime.
- Especially when debugging, I added a lot of print statements to view all information about the instructions in each pipeline, **EVERY SINGLE CYCLE**. I also drew Anyhow, doing this helps me guarantee that the program is working like how I expected, especially the special cases (including priority of issue, CDB, …) and also the cycle between when a value is broadcasted and next instructions can be issued.

4. BUGs:

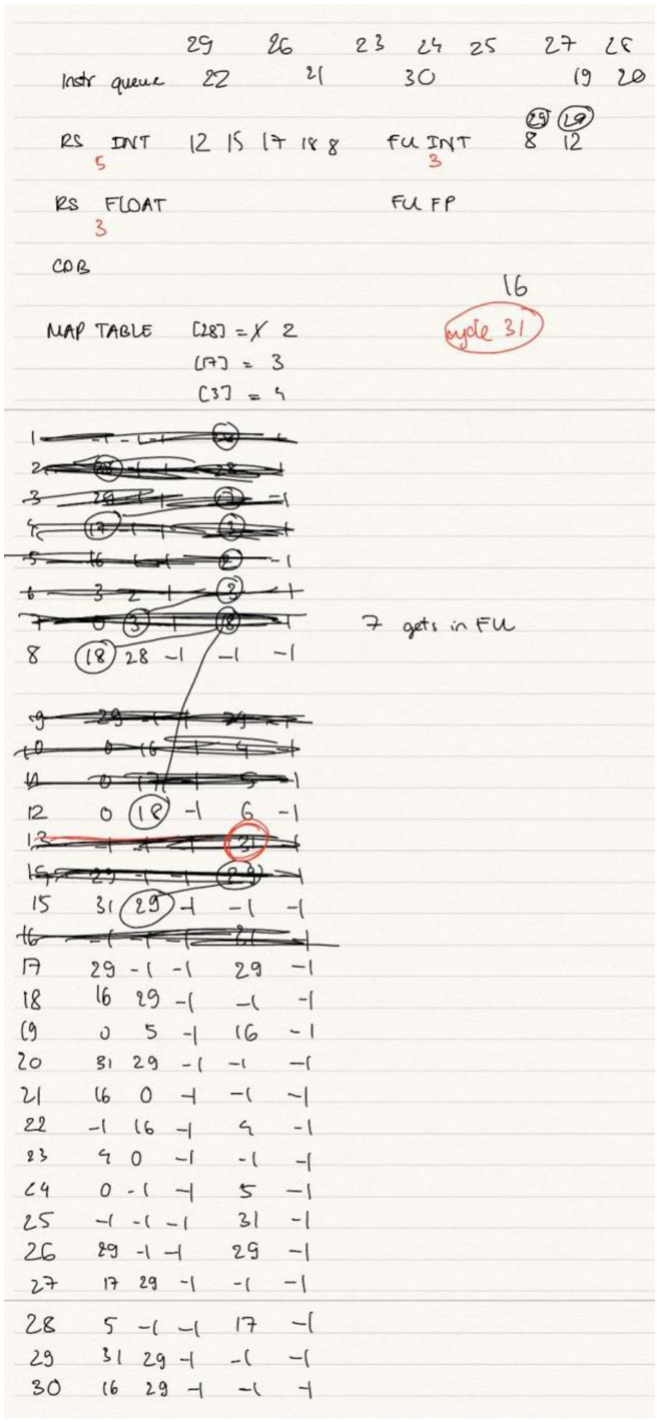
- First bug I had was infinite loop program. This is because I assumed that an instruction will not have 2 out register values, so I added a “break” statement while checking tags for map_table.
- The second tough one I had was to notice that conditional statements can still have out_reg different from -1. Therefore, without an if check in dispatch_to_issue (line 483), an out_reg will never be broadcasted, leading to other instructions using the same map_table entry will **stuck** since younger instructions will simply wait for the tag to broadcast. Cost me 3 hours to find this out.



```
// If a hazard was not detected, remove the instruction from the IFQ
// This means the instruction has ALREADY been moved to a reservation station
if (IS_UNCOND_CTRL(instr->op) || IS_COND_CTRL(instr->op)) {
  // Do not care about register values
}
else {
  if (instr->r_in[0] > 0) instr->Q[0] = map_table[instr->r_in[0]];
  if (instr->r_in[1] > 0) instr->Q[1] = map_table[instr->r_in[1]];
  if (instr->r_in[2] > 0) instr->Q[2] = map_table[instr->r_in[2]];
  if (instr->r_out[0] > 0) map_table[instr->r_out[0]] = instr;
  if (instr->r_out[1] > 0) map_table[instr->r_out[1]] = instr;
}
instr_queue[instr_queue_size-1] = NULL;
instr_queue_size--;
```

5. **Distribution**: My teammate dropped the course so I did everything on my own

Bonus: How I figured out the second bug



29 26 23 24 25 27 28

Instr queue 22 21 30 19 20

RS INT 12 15 17 18 8 FU INT 8 12

5 3

29 29

RS FLOAT FU FP

3

COB

16

MAP TABLE [28] = ~~X~~ 2 cycle 31

[17] = 3

[3] = 4

7 gets in FU

~~1~~	~~-1~~	~~-1~~	~~-1~~	~~28~~	~~-1~~
~~2~~	~~28~~	~~-1~~	~~-1~~	~~28~~	~~-1~~
~~3~~	~~28~~	~~-1~~	~~-1~~	~~3~~	~~-1~~
~~4~~	~~17~~	~~-1~~	~~-1~~	~~3~~	~~-1~~
~~5~~	~~16~~	~~-1~~	~~-1~~	~~2~~	~~-1~~
~~6~~	~~3~~	~~2~~	~~-1~~	~~3~~	~~-1~~
~~7~~	~~0~~	~~3~~	~~-1~~	~~18~~	~~-1~~
8	18	28	-1	-1	-1
~~9~~	~~29~~	~~-1~~	~~-1~~	~~24~~	~~-1~~
~~10~~	~~0~~	~~16~~	~~-1~~	~~4~~	~~-1~~
~~11~~	~~0~~	~~17~~	~~-1~~	~~5~~	~~-1~~
12	0	18	-1	6	-1
~~13~~	~~-1~~	~~-1~~	~~-1~~	~~31~~	~~-1~~
~~14~~	~~29~~	~~-1~~	~~-1~~	~~29~~	~~-1~~
15	31	29	-1	-1	-1
~~16~~	~~-1~~	~~-1~~	~~-1~~	~~31~~	~~-1~~
17	29	-1	-1	29	-1
18	16	29	-1	-1	-1
19	0	5	-1	16	-1
20	31	29	-1	-1	-1
21	16	0	-1	-1	-1
22	-1	16	-1	4	-1
23	4	0	-1	-1	-1
24	0	-1	-1	5	-1
25	-1	-1	-1	31	-1
26	29	-1	-1	29	-1
27	17	29	-1	-1	-1
28	5	-1	-1	17	-1
29	31	29	-1	-1	-1
30	16	29	-1	-1	-1