# Lab 3 report

Yue Huang 1003839484
Jiayi Zhao 1003936592

1.

For the first one million instructions of the three traces, we got
1814116 total number of cycles with tomasulo for gcc.eio
1852941 total number of cycles with tomasulo for go.eio
1979817 total number of cycles with tomasulo for compress.eio

2.

we called the functions in the while loop in the following order:

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

CDB_To_retire corresponds to Writeback stage, execute_To_CDB corresponds to execute stage, issue_To_execute corresponds to issue stage, dispatch_To_issue corresponds dispatch stage, fetch to dispatch corresponds to fetch stage. Because fetch and dispatch is merged into one stage in this lab, we swapped the order of these two functions to make the two stages happen in one cycle.

In fetch_To_dispatch, we first fetch the instructions into the Instruction Fetch Queue(IFQ), in fetch function, we have to consider the case of TRAP instruction and skip it so it will not be fetched into the IFQ. The instructions are fetched into the IFQ until it is full. After each instruction is fetched, the tom_dispatch_cycle of each instrction is set to current cycle.

In dispatch_To_issue, the instructions are dispatched under three conditions.

1. If the instruction is an unconditional branch or conditional branch, it is removed from the head of the IFQ and finishes.
2. If it is an instruction that will use integer functional unit, we first check any of its source operand is dependent on other previous instructions from the map table and set tags to Q if there is dependency. Then we update the tags of map table corresponding to the destination of the instruction. Finally, we reserve an integer reservation station for this instruction and set tom_issue_cycle to be the next cycle.
3. If it is an instruction that will use floating point functional unit, we do the similar operations as the instructions that use integer functional unit.

A helper function update_instr_queue is used. Every time an instruction is dispatched, we remove it from the IFQ.

In issue_To_execute, we first check all the instructions in the reservation stations whether any of the instructions list Q is resolved (all NULL), which means it is ready to execute. If list Q is all NULL, we give it a ready state. We choose two oldest integer instructions and one oldest floating-point

instructions among all the ready to execute instructions. We then check if there is any free space in corresponding functional unit, if there is, we assign the oldest instructions to the functional unit and set the tom_execute_cycle to next cycle.
In this stage, one special concern is that we have two integer functional unit. Every cycle we choose, if there are any, two oldest integer instructions that can be executed because both of the functional unit may be free to use at next cycle. Then, we check if there are spaces available in the integer functional unit. If there is none, wait till next cycle. If there is only one, let the oldest execute. If there are two, assign both of them to integer functional unit. We have a helper functional called insert. Every we call insert we insert an instruction that is ready to execute into an ready array. Inside insert we check whether the number of the ready instructions is more than 2 and whether this instruction is older than any other instruction already in the array. The output of insert is the two oldest ready instructions kept in the array for that time.

In execute_To_CDB, we first increase the cycle count that every instruction has stayed in the functional unit. If any instruction is finished in this cycle, it is ready to be broadcasted by CDB. We choose the oldest instruction to take CDB next cycle, other instructions have to be stalled. In this stage, we have to choose the oldest among integer functional unit and floating point functional unit. Finally, we free the functional unit and reservation station of the oldest instruction.
A special case is if a store instruction is finished, it does not take the CDB, it just free its functional unit and reservation station once it is finished.

In CDB_To_Retire, an instruction is broadcasted through CDB. We first go through the instructions both in integer and floating point reservation station. If any Q tags matches CDB, set it to NULL, meaning dependency is resolved. Finally, we go through the map table, if any tag matches CDB instruction, set it to NULL, meaning the value is written to register files.

In is_simulation_done, we check if there are any instructions in integer and floating point reservation stations and functional units. If there are no instructions, meaning the simulation is done.

3.
To test our program, we used print_all_instr function to print out the cycle diagram. We printed out 10/20 instructions and check whether the cycle stage matches what we expected.

4.
At first, we didn't notice fetch_index is an instruction index last fetched. We used it as index to be fetched, thus the first instruction we fetched in at index 0. And because of this, our instructions in stuck in the IFQ and we had an infinite loop in dispatch_to_issue. We used gbd to check what is going on and found that instruction 0 has opcode of 0. So, it will be stuck in the queue forever. We then simply changed the initial value of fetch_index from 0 to 1.

There were a lot of segmentation fault errors when we first try to run the program. We then used gdb to trace it. But every time one error is fixed, another one may pop up. Because we didn't

include some of the corner cases, causing the program to get instruction information (i.e. index or op) from NULL.

5.Both of us did the coding part individually. Yue did the report.