Ringo Cheung - 1000463955
Yassir Solomah - 1000458284

Lab 3 - Dynamic Scheduling with Tomasulo

1 Results

EIO trace	Total number of cycles with Tomasulo
gcc	1 814 118
go	1 852 943
compress	1 979 819

Table 1: Total number of cycles with Tomasulo for different EIO benchmarks

2 Code

2.1 Fetch

First, check whether we can still fetch instructions. We can only fetch instructions if there are still instructions in the simulation left and the instruction fetch queue (IFQ) is not full. Then keep fetching until we get an instruction that is neither a trap nor a nop. Finally, put the fetched instruction into the instruction fetch queue and set its dispatch cycle to the current cycle.

2.2 Fetch to Dispatch

Call fetch since it takes care of the dispatch as well.

2.3 Dispatch to Issue

Pop the instruction at the top of the queue. Decode the instruction. If the instruction is a branch then do nothing. If the instruction uses the floating point or integer functional units, look for a free reservation station and push the instruction to the reservation station using the pushToReservation (2.11) helper function. If there are no free reservation stations then push the instruction back to the top of the queue as instructions must be dispatched from the instruction queue in order.

2.4 Issue to Execute

For each integer functional unit, attempt to execute the oldest unexecuted instruction in a reservation station that uses an integer functional unit by calling issue_oldest_To_execute_INT (2.9). Do the same for the floating point functional units using issue_oldest_To_execute_FP (2.9).

2.5 Execute to CDB

Iterate through the integer and floating point functional units looking for the oldest completed instruction. During this search, if we find any instructions that have finished executing and do not write to the cdb, remove them from their reservation stations and functional units. If an instruction exists that is finished executing and is the oldest, place the value it produced onto the common data bus, set its cdb cycle to the current cycle and free the reservation station and functional unit that it was using.

2.6 CDB to Retire

Check if the common data bus has a value written to it. If so, broadcast to all the instructions in reservation stations and set their sources to NULL if the source register name matches the common data bus's instruction output register name. Clear the map table of the common data bus's instruction output registers. Clear the common data bus.

2.7 Remove from functional unit

Takes the instruction to remove as an argument. Iterate through the functional units and if the instruction matches the one in the functional unit, remove the instruction from the functional unit by setting the functional unit to NULL.

2.8 Remove from reservation station

Takes the instruction to remove as an argument. Iterate through the reservation stations and if the instruction matches the one in the reservation station, remove the instruction from the reservation station.

2.9 Issue oldest to execute FP/INT

Iterate through the reservation stations looking for the oldest instruction that has not yet been executed. Assign it to a functional unit if one is free. If the oldest instruction was assigned to a functional unit, set its execute cycle to the current cycle.

2.10 Check dependency

Takes an instruction as an input. Returns true if all the source registers Q are NULL and false otherwise.

2.11 Push to reservation

Takes as inputs an instruction, a pointer to the reservation table, an index and the current cycle. Iterate through the input registers of the instruction and set them to the instructions that will produce their values according to the map table. Iterate throught the output registers of the instruction and write them to the map table. Set the reservation table at the index to the current instruction. Set the issue cycle to the current cycle.

3 Test for correctness

In order to test for the correctness of the code, we printed out the first 10 instructions using the trace by calling `sim-safe -max:inst 10 /cad2/ece552f/benchmarks/gcc.eio`. The instruction trace is printed out below:

```
TOMASULO TABLE
(1)  lw       r16,0(r29)         1      2      3      7
(2)  lui      r28,0x1003         2      3      4      8
(3)  addiu    r28,r28,20912      3      4      9      13
(4)  addiu    r17,r29,4          4      5      7      11
(5)  addiu    r3,r17,4           5      7      13     17
(6)  sll      r2,r16,2           6      8      11     15
(7)  addu     r3,r3,r2           7      11     18     22
(8)  addu     r18,r0,r3          8      13     23     27
(9)  sw       r18,-21500(r28)    9      15     28     0
(10) addiu    r29,r29,-24        10     17     18     23
```

By going through the trace, we can see that each instruction is going through each stage when it is supposed to. We repeated this procedure for up to 40 instructions for which the trace is not in this report due to space constraints. For example, we can see that instruction 4 executes before instruction 3 because instruction 3 has a RAW dependency on 2 that it must wait for but instruction 4 does not have any dependencies. Furthermore, instruction 4 starts executing in the same cycle that instruction 1 finished executing and not any sooner because there are only two integer functional units and they are both being used by instructions 1 and 2. We went through more instructions like this to verify that the simulator works as intended.

4 Difficulties

Throughout this practicum, we ensured the input trace was being correctly simulated by printing subsets of the *Tomasulo Table*, and comparing that to a hand-completed table.

4.1 Bug one

When completing the *execute_to_CDB*, we were unsure of whether would we could remove an instruction from the reservation after being pushed into a functional unit. This arose after believing that there are no longer anymore dependencies at this point, and the reservation station is unnecessary to the operation of the functional unit.

In short, we came to the conclusion that the reservation needs to remain with the functional unit as the operands are not copied from the reservation station into the functional unit. In our current implementation, we only remove from the reservation station and functional unit provided it's the oldest finished instruction or does not write to CDB.

```
void execute_To_CDB(int current_cycle) {
    instruction_t * oldestFinished = NULL;
    ... // 1. Free the RS & FU for Finished & !Writes_cdb
    ... // 2. Find oldest finished instruction requiring CDB

    // Oldest Completed instruction to be moved to CDB and removed from RS & FU
    if(oldestFinished &&  !commonDataBus) {
     ... // Assign to the CDB, remove from RS & FU, set cycle
    }
}
```

4.2 Bug two

Another bug we ran into was related to the *issue_To_execute*, as we were unsure of how many instructions could be pushed into execution during a cycle. After reviewing the Tomasulo algorithm, it was clear we could push in as many instructions as possible.

After passing this point, we didn't think of the modularity of writing code that could support N integer functional units and M floating point functional units. This resulted in the section being messy and bloated; thus, difficult to debug. We concluded that efficiency of the simulation was not main effect of the coding, so we separated the *issue_To_execute* into two for loops:

```
void issue_To_execute(int current_cycle) {
    for(int i = 0; i < FU_INT_SIZE; i++) {
     issue_oldest_To_execute_INT(current_cycle);
    }
    for(int i = 0; i < FU_FP_size; i++) {
     issue_oldest_To_execute_FP(current_cycle);
    }
}
```

Thus, this allowed us to accept arbitrary number of function units, and created modularity for debugging ease.

5 Statement of Work

Equal work was completed by both partners.