ECE 552 Lab 1 Report

Francis Guo 1005931397 Rongbo Zhang 1005273393

For the total numbers of cycles with Tomasulo for the first 1M instructions of each EIO trace:

Traces	gcc.eio	go.eio	compress.eio
Tomasulo cycle	1,681,443	1,695,064	1,851,550

Table 1: Tomasulo simulation results

For the algorithm of the Tomasulo, here is how we achieve the Tomasulo processor:

- **Is_simulation_done**
  We check whether there are more instructions that are not executed by checking if all pointer values are NULL in IFQ, Map Table (MP), Reservation Stations (RS), Functional Units (FU), CDB. Once all of the mentioned positions are emptied, the simulation is finished.
- **CDB_To_retire**
  If the CDB is used, then clean the corresponding map table with a tag match to indicate the value is available in regfiles.
- **execute_To_CDB**
  For the execution to CDB, we go through all the functional units, to check if any instruction finished the execution and ready to send the data to the CDB. To resolve the structural hazard, we keep track of the oldest instruction found when iterating through both FP and INT functional units, and only send the oldest instruction to CDB if it is available and clear its FU and RS entries. The reason we can clear the RS here is that the CDB gets the tag here, so the RS entries can be cleared.
- **issue_To_execute**
  For the execution, we iterate through the INT-RS and FP-RS separately to get the oldest instructions that have their tags cleared. Then we check if the corresponding FUs are available. Then we issue the eldest instruction to the corresponding empty function unit. If there is no function unit free, the instructions in the reservation station need to wait for the function unit to be free. After we have done all these checks, we can update tags in MT and RS according to the CDB broadcast. This is because, as mentioned in the handout, the instruction needs to wait until the next cycle after CBD broadcast when issuing to the execution stage.
- **dispatch_To_issue**
  We check if there are any RS entries that are free. we pop the instruction from IFQ and add it to the corresponding RS if it needs any. When an instruction is issued, check if there is data dependence, and set the tag according to the map table. And update the map table for the output register.
- **fetch_To_dispatch**
  Since fetch and dispatch are merged, this function assigns the start dispatch cycle number to the corresponding instruction that is fetched in this cycle.
- **fetch**
  If IFQ is not full then extract the next non-trap instruction from the trace to IFQ
- **helper_functions:**
  - rm_from_RS: remove the instruction from the corresponding RS entry.
  - broadcast_tags: clear the tags in MT and RS if there is a tag match.
  - set_tags: set the output tag in MT and input tag in RS
  - IFQ_pop_front: pop the first instruction from IFQ and shift the queue to left by 1

For the testing of the code, we print out the first 30 instructions. Then based on the print out information, We used hand progression to find out the cycle count for each instruction and each stage. We need to check if the dispatch cycle, issue cycle and execute cycle are processed the same as the hand calculation. After the first 30 instructions are correct, we could keep increasing the instruction printing count and verify the next instruction has the correct cycle count for each stage. Until we reach the simulation requirement. We end up checking 1000 instructions in the gcc EIO

trace to make sure all instructions and the special cases are handled correctly. such as RAW hazards and structural hazards need to stall for the dependent data and function unit to be free.

TOMASULO TABLE

lw	r16,0(r29)	1	2	3	8	
lui	r28,0x1003	2	3	4	9	
addiu	r28,r28,20912	3	4	10	15	
addiu	r17,r29,4	4	5	6	11	
addiu	r3,r17,4	5	6	12	17	
sll	r2,r16,2	6	8	9	14	
addu	r3,r3,r2	7	9	18	23	
addu	r18,r0,r3	8	11	24	29	
sw	r18,-21500(r28)		9	14	30	0
addiu	r29,r29,-24	10	15	16	21	
addu	r4,r0,r16	11	17	18	24	
addu	r5,r0,r17	12	21	22	27	
addu	r6,r0,r18	13	23	30	35	
jal	0x5fb810	14	0	0	0	
addiu	r29,r29,-24	15	25	26	31	
sw	r31,16(r29)	16	27	32	0	
jal	0x602f70	17	0	0	0	
addiu	r29,r29,-24	18	29	35	40	
sw	r16,16(r29)	19	31	41	0	
addu	r16,r0,r5	20	35	36	41	
sw	r31,20(r29)	21	36	41	0	
beq	r16,r0,0x602fe0		22	0	0	0
lw	r4,0(r16)	23	38	46	51	
beq	r4,r0,0x602fe0		24	0	0	0
addiu	r5,r0,47	25	40	41	46	
jal	0x6089a0	26	0	0	0	
addiu	r29,r29,-32	27	42	46	52	
sw	r17,20(r29)	28	46	53	0	
andi	r17,r5,255	29	47	48	53	
sw	r31,24(r29)	30	48	53	0	

Figure 1: the first 30 instruction output of the gcc.eio trace

For the problem we have during the debug process.

1. The first issue we encountered during the development of the Tomasulo code is the instruction occasionally stuck in the reservation station. We had to go through the print out information of the reservation station and maptable. The log is showing there is a mishandling of the RAW hazard. So, we go back to realize that the map table tag showing the data dependency is updated before we check the data dependency in the reservation station. When the instruction writes back to the same register that it is dependent on and the previous tag in the map table is updated in the same cycle as it starts to issue, then its tag is mis-labeled to depend on its output. We could only find this issue on a few instructions in 100 instructions, so it was not hard to realize this corner case. To resolve it we reposition the

data dependence resolve check after the reservation station check which instruction could be processed to execution which solves the problem.

2. The second issue we encountered was that the instruction started to execute under an unexpected clock cycle which was one cycle ahead before we expected it to start to execute and getting stuck in the functional unit. We had to go through the print out information of the reservation station and function unit. Because we adopt the structure of reverse execution flow of the pipeline to make sure the instruction flow as expected. Which means the actual steps are done in the order of CDB_retire→ execute_To_CDB →issue_To_execute→dispatch_To_issue→fetch_To_dispatch. This means the original order of updating the reservation data dependency tags right after CDB broadcast is incorrect because we check the data dependence after the CDB broadcast. So, we have to move the clean reservation station after we check the data dependence tags in the reservation station.

Work Breakdown:

Rongbo Zhang:

- Implemented functions execute_To_CDB, issue_To_execute, dispatch_To_issue, fetch, fetch_To_dispatch, and helper functions.
- debugging using gdb for segmentation faults

Francis Guo:

- Tomasello processor debug for the clock matching the calculation.
- Implemented function CDB_To_retire, is_simulation_done
- debugging using printout for cycle dismatch