Report the “total numbers of cycles with tomasulo” for the first one million instructions of each EIO trace.

benchmark	#cycle
gcc	1681443
go	1695064
compress	1851550

Briefly describe your code for each Tomasulo stage (i.e., provided function) at an algorithmic level. Make sure to point out any cases that required special handling. Also include high-level descriptions of any significant helper functions you wrote.

dispatch_To_issue

```
void dispatch_To_issue() {
  branches are not dispatched, skip them
  call get_free_reserv_station_entry() from corresponding reserv_station ba
sed on the op code
  pop_insn_queue()
  update_insn_Q_and_map_table(insn)
}
```

Helper function:

`get_free_reserv_station_entry`: Get a free reservation station entry for `rsv_station_entry` instruction

`update_insn_Q_and_map_table`: update `map_table[insn->r_out[i]]` and `insn->Q[i]`

`pop_insn_queue`: Pop first instruction from the insn queue, and shift all others forward by one position

issue_To_execute

```
void issue_To_execute() {
  copy_ready_insn_from_reserv_station_to_fu(INT, current_cycle);
  copy_ready_insn_from_reserv_station_to_fu(FP, current_cycle);
}
```

Helper function

```
void copy_ready_insn_from_reserv_station_to_fu() {
  find a free functional unit
  find the ready instruction with minimum index
  copy the ready instruction from reservation station to functional unit
}
```

`find_free_fu`: Find a free functional unit

`find_ready_reserv_station_entry_w_min_idx`: Get the ready instruction with minimum index. To check if the instruction is ready, we have to check RAW dependencies using `has_raw_dependences()`

`has_raw_dependences`: Check if `insn` has RAW dependences

execute_To_CDB

```
void execute_To_CDB() {
  broadcast_insn = find_completed_fu_with_min_index()
  commonDataBus = broadcast_insn
}
```

Helper function:

```
functional_unit_entry_t* find_completed_fu_with_min_index() {
  for all entries in both INT and FP fu
    check if insn's complete_cycle <= current_cycle, if so
      check IS_STORE(insn), if so
        free insn's reservation station and fu resource, as store is not
going to cdb stage
      else
        find one instruction with minimum index
  return found insn
}
```

CDB_To_retire

```
void CDB_To_retire() {
  free_reserv_and_fu_resource_at_cdb()
  keep track of last_cdb and last_cdb_address for later update_map_table_w_
cdb() and update_reserv_station_w_cdb() calls
}
```

Helper function:

`update_map_table_w_cdb`: Update `map_table` with `cdb` instruction. Mark the output register of instruction `cdb` as `NULL` (ready to use)

`update_reserv_station_w_cdb`: Update insns in reservation station that is waiting for `cdb` for its input register. Mark them as ready

`free_reserv_and_fu_resource_at_cdb`: Free the resource of `cdb` instruction in reservation station and functional unit

Notice the other two functions associate with CDB stage `update_map_table_w_cdb()` and `update_reserv_station_w_cdb()` are not done in `CDB_To_retire()`. But at the end of the `while` loop in `runTomasulo`. Because according to the lab handout, if insn A is waiting for a value from insn B, if insn B is in Writeback on cycle 9, then A can enter Execute on cycle 10. So we can't update map_table and reserv_station, otherwise the freed resource will be use directly by `execute_To_CDB()`.

```
counter_t runTomasulo(){
  while (true) {
    CDB_To_retire(cycle);
    execute_To_CDB(cycle);
    issue_To_execute(cycle);
    dispatch_To_issue(cycle);
    fetch_To_dispatch(trace, cycle);

    update_map_table_w_cdb()
    update_reserv_station_w_cdb() for both INT and FP servation station
}
```

Explain how you tested the correctness of your code

I use the `compress.eio` benmark and here's the first 13 instructions. I marked dependencies using the same colors and also go over the structural dependencies instruction after instruction.

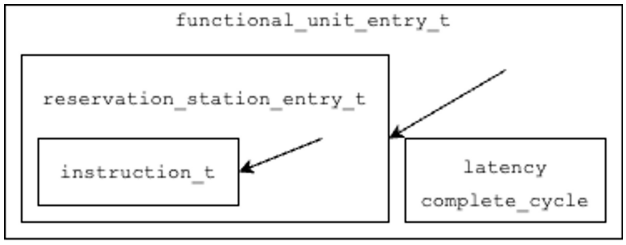
NO	Insn	D	S	X_s	W (CDB)
1	`lw r16,0(r29)`	1	2	3	8
2	`lui r28,0x1003`	2	3	4	9
3	`addiu r28,r28,20912`	3	4	10	15
4	`addiu r17,r29,4`	4	5	6	11
5	`addiu r3,r17,4`	5	6	12	17
6	`sll r2,r16,2`	6	8	9	14
7	`addu r3,r3,r2`	7	9	18	23
8	`addu r18,r0,r3`	8	11	24	29
9	`sw r18,-21500(r28)`	9	14	30	0
10	`addiu r29,r29,-24`	10	15	16	21
11	`addu r4,r0,r16`	11	17	18	24
12	`addu r5,r0,r17`	12	21	22	27
13	`addu r6,r0,r18`	13	23	30	35

Briefly describe the two toughest bugs you had while developing your Tomasulo code

The first one is in function `update_insn_Q_and_map_table`. I updated the output first and then input. But this gives me circular dependencies, resulting in instructions waiting forever.

```
instruction_t* update_insn_Q_and_map_table(instruction_t* insn) {
    //* Must update `insn` input first, otherwise `insn` that has same i
nput and output register will be have circular dependency
    for (int i = 0; i < 3; i++) {
        if (insn->r_in[i] != DNA) {
            insn->Q[i] = map_table[insn->r_in[i]];
        }
    }

    // Update `insn` output
    for (int i = 0; i < 2; i++) {
        if (insn->r_out[i] != DNA) {
            map_table[insn->r_out[i]] = insn;
        }
    }
    return insn;
}
```



The second bug is related to the design of data structure. Deeper stage requires information of previous stage. E.g. Functional unit, which need information of reservation station when we free the resource in the CDB stage; Reservation station entry need information of the instruction. At first the `functional_unit_entry_t` structure doesn't include pointer to `reservation_station_entry_t` . Thus, freeing reservation station resource is cumbersome and causes bugs that are hard to debug. We change the design to solve this.

Include a brief statement of work completed by each partner

We evenly distribute our work. We think about the general design together. Yuhe did most of the coding and Rudy Jin did more testing and debugging.