1. How can the Tomasulo algorithm extract more ILP for a given program compared to an in-order processor?

Tomasulo algorithm extract more ILP by enabling out of order execution. In an in-order processor, when there is a RAW hazard, CPU stalls current execution and disallowing following instructions to execute even if there are no hazards. Tomasulo algorithm enables those instructions to execute by dispatching them into the reservation stations. And issuing them whenever there are no RAW hazards.

2. What are reservation stations used for? List all the fields of a reservation station entry and explain their functionality.

Reservation station is used to store the instruction's opcode, its functional unit, its operands tags or values if they are ready. A reservation station consists of:

FU: Functional unit ID

Busy: whether this reservation station is occupied

Op: the instruction's op code

Qj, Qk, …: source register tags (RS# of RS) that will produce the input value

Vj, Vk, …: source register value

3. How does Tomasulo's algorithm deal with false dependences? Provide a brief example.

Tomasulo's algorithm deals with false dependencies by copying the source register values or the tag that produces the values into RS and keeping track of the source register tag (RS# of RS). On a writeback, CDB broadcasts to RS and map table and updates all the register value that matches with the tag. (This is also known as register renaming)

Example:

I1: div r2, r3 -> r1

I2: add r2, r1 -> r3

I3: subi r4, 4 -> r1

RAW dependencies: I1 and I2 (r1)

WAW dependencies: I1 and I3 (r1)

WAR dependencies: I1 and I2 (r3), I2 and I3 (r1)

If you use register renaming in this example, the instructions become:

I1: div p2, p3 -> p5

I2: add p2, p5 -> p6

I3: subi p4, 4 -> p7

The WAW and WAR dependencies are removed, and only RAW dependencies remain.