**PRELAB**

**Lab - 3**

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

Tomasulo’s algorithm is a dynamic scheduling algorithm that allows out-of-order execution and enables more efficient use of multiple execution units. Here, instructions are able to start their execution in any order that respects their dependencies, but not necessarily in the process order. Since the process order in not maintained, “waits” are used instead of stalls, during “wait” younger instructions may still proceed with execution. This increases the ILP. Whereas in static architectures an instruction does not start execution until it is free of all hazards with previous instructions in process order, due to this stalls are introduced which limits ILP.

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

Reservation stations are used for register renaming in the Tomasulo’s algorithm. They permit the CPU to fetch a data value as soon as it has been computed. When instructions are issued, they can designate the reservation station from which they want their input to read.

Fields of a reservation station-

-> FU - Each functional unit has a single reservation station. Reservation stations hold information needed to execute a single instruction. The unit begins processing when it is free and when all source operands needed for an instruction are real.

-> Busy Flag - This notifies if the FU is free or not.

-> OP - This holds the opcode in it.

-> Qj, Qk - These point towards the operands which are not yet ready.

-> Vj, Vk - These point towards the operands which are ready. If an operand is in either Qj, Qk and they get ready, they are moved to corresponding Vj or Vk.

1. **How does Tomasulo's algorithm deal with false-dependencies? Provide a brief example.**

Tomasulo’s algorithm deals with false-dependencies by using Register renaming. It is a technique that eliminates the false data dependencies arising from the reuse of architectural registers by successive instructions that do not have any real data dependencies between them.

Example,

LD 0 (r2) -> r1

ADD r2, 50 -> r1

ST r1 -> r3

ADD r2, 90 -> r1

…

…

Solution,

LD 0 (r2) -> r1

ADD r2, 50 -> r1

ST r1 -> r3

ADD r2, 90 -> r4

…

…