→ Tomasulo's algorithm

RAW hazards are removed by executing an instruction only when its operands are available. WAR and WAW hazards are eliminated by register renaming.

Register renaming is provided by reservation stations, which buffer the operands of instructions waiting to be executed. The basic idea is that a reservation station fetches and buffers an operand as soon as it's available, eliminating the need to get the operand from a register of the register bank.

These reservation stations lead to two important properties:

- hazard detection and hazard control are distributed
- results are passed directly to the functional units from reservation stations where they are buffered, rather than going through the registers of the register bank. (forwarding)

Also, it features distributed RS: operands are forwarded to multiple RSs instead of using a centralized register bank.

It also supports in-order issue but out-of-order instruction completion.

↳ Tomasulo Pipeline Phases

- IF: fetch next instr. into FIFO of pending instructions
- Issue: get next instr. from head of queue. If a

a matching reservation station is free, the instr. is issued to the station with the operand values if they are currently saved in registers of the register bank. Otherwise, tracking of the functional units which will produce these operands is performed.

- **Execute:** when all operands are available (no RAW) and FU is free, execute. If P reservation stations may be chosen arbitrarily. If not, the reservation station monitors the common data bus for the result to become available.
- **Write the result:** The FU writes the result to the CDB (to all the reservation stations waiting for it) and to the register bank. Also, the reservation station is marked as free.

↳ How load and store are executed:

- load and store consists of 2 steps:
  - effective address (EA) calculation
  - memory access
- load and store buffers each have a field A (address) which is calculated in step 1 (EA).
- Store buffer Val field can be a RS producing this value.
- A RS can be waiting for a load buffer.

→ **Load:** - Address unit calculates EA.
- EA is compared to A field of all active store buffers. If there is a match, we are

in a RAW and load is not sent to the load buffer until conflicting store completes.

→ store: Similar but must check for conflicts in both load buffers (WAR) and store buffers (WAW).

→ Hardware-based speculation

With speculation, instructions are fetched, issued and executed as if branche predictions were always right. Dynamic scheduling only fetches and issues such instructions. Of course, mechanisms are needed to handle the situation where speculation turns out to be incorrect

HW-based speculation combines 3 ideas:

- dynamic branch prediction (to predict)
- speculation, to allow the execution of instructions before the control dependences are resolved.
- dynamic scheduling to deal with the issuing of different combinations of these blocks.

Instructions can execute out-of-order but are forced to commit in-order. This separation is essential because instructions may finish executing before they are ready to commit.

Additional hardware is used to store the results of completed instructions that have not committed yet - reorder buffer ROB - which is also used to pass results between instr. Speculative instructions are not allowed to change

Speculative processor state (memory, registers, ...)

↳ ROB structure

- **busy**: whether entry is occupied or free.
- **type**: type of instruction (branch, load, store or ALU)
- **destination location**: destination registers or memory addr.
- **value**: value of the instruction result (between completion and commitment)
- **ready**: if instruction has completed execution.

↳ Instruction phases using ROB

1- Issue: next instruction is obtained from the head of the instruction queue. If RS and ROB slot are free, the instr. is issued to the station together with the operands values, if they are currently saved in registers or ROB entries.

2. Execute: CBD is monitored while waiting for the operands to be computed (prevent RAW). When operands are available, the operation is executed.

3. Write result: when instr. completes, its written into CBD with the ROB entry tag, to be written into the ROB and all RS. (Store → write the value to Value field of store's ROB entry. If no value available, monitor CBD).

4. commit: instruction attains the head of ROB and ready is set, the processor updates the register of the register bank and finishes. For a branch with wrong prediction, the ROB is flushed and fetching is restarted at the proper address. Store writes to memory.

↳ Pros and cons

Advantages: - Use FUs that would otherwise be unused.
- eliminate control hazard stalls.

Potential disadvantages - consumes time and energy of wrongly speculated.
- requires HW and power.
- performance reduction if speculative instr. causes exceptional event. (cache miss, ...)