Tomasulo’s algorithm

- “Weaknesses” in scoreboard:
  - Centralized control
  - No forwarding (more RAW than needed)
- Tomasulo’s algorithm as implemented first in IBM 360/91
  - Control decentralized at each functional unit
  - Forwarding
  - Concept and implementation of renaming registers that eliminates WAR and WAW hazards

Tomasulo’s solution to resolve hazards

- Structural hazards
  - No free reservation station (stall at issue time), No further issue
- RAW dependency (detected in each functional unit -- decentralized)
  - The instruction with the dependency is issued (put in a reservation station) but not dispatched (stalled). Subsequent instructions can be issued, dispatched, executed and completed.
- No WAR and WAW hazards
  - Because of register renaming through reservation stations
- Forwarding
  - Done at end of execution by use of a common (broadcast) data bus

Example machine (cf. Figure 3.2)

An instruction goes through 3 steps

- Assume the instruction has been fetched
- 1. Issue, dispatch, and read operands
  - Check for structural hazard (no free reservation station or no free load-store buffer for a memory operation). If there is structural hazard, stall until it is not present any longer
  - Reserve the next reservation station
  - Read source operands
    - If they have values, put the values in the reservation station
    - If they have names, store their names in the reservation station
  - Rename result register with the name of the reservation station in the functional unit that will compute it

An instruction goes through 3 steps (c’ed)

- 2. Execute
  - If any of the source operands is not ready (i.e., the reservation station holds at least one name), monitor the bus for broadcast
  - When both operands have values, execute
- 3. Write result
  - Broadcast name of the unit and value computed. Any reservation station/result register with that name grabs the value
- Note two more sources of structural hazard:
  - Two reservation stations in the same functional unit are ready to execute in the same cycle; choose the “first” one
  - Two functional units want to broadcast at the same time. Priority is encoded in the hardware
Implementation

- All registers (except load buffers) contain a pair
  \{value, tag\}

- The tag (or name) can be:
  - Zero (or a special pattern) meaning that the value is indeed a value
  - The name of a load buffer
  - The name of a reservation station within a functional unit

- A reservation station consists of:
  - The operation to be performed
  - 2 pairs (value, tag) \( (V_j, Q_j), (V_k, Q_k) \)
  - A flag indicating whether the accompanying f-u is busy or not

\[ \text{Name} \quad \text{Busy} \quad \text{Fm} \quad \text{Vj} \quad \text{Vk} \quad \text{Qj} \quad \text{Qk} \]

\[ \begin{array}{cccccc}
\text{Add F6,F8,F2} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\text{Div F10, F0, F6} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\text{Sub F8, F6, F2} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\text{Mul F0, F2, F4} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\text{Load F2, 45(r3)} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\text{Load F6, 34(r2)} & \text{yes} & \text{yes} & \text{yes} & \text{yes} \\
\end{array} \]

\[ \begin{array}{cccccc}
\text{Name} \quad \text{Busy} \quad \text{Fm} \quad \text{Vj} \quad \text{Vk} \quad \text{Qj} \quad \text{Qk} \\
\text{Add 1} & \text{yes} & \text{Sub (Load1)} & 0 & \text{Load2} & \text{Qj} \Rightarrow 0 \text{ means Vj has a value} \\
\text{Add 2} & \text{yes} & \text{Add (Load2)} & 0 & \text{Add1} & \text{Load2} \\
\text{Mul 1} & \text{yes} & \text{Mul (Load 2)} & \\
\text{Mul 2} & \text{yes} & \text{Div (Load 1)} & & 0 & 0 \\
\end{array} \]

- The tag (or name) can be:
  - Zero (or a special pattern) meaning that the value is indeed a value
  - The name of a load buffer
  - The name of a reservation station within a functional unit

- A reservation station consists of:
  - The operation to be performed
  - 2 pairs (value, tag) \( (V_j, Q_j), (V_k, Q_k) \)
  - A flag indicating whether the accompanying f-u is busy or not

\[ \text{Instruction} \quad \text{Issue} \quad \text{Execute} \quad \text{Write result} \]

\[ \begin{array}{cccc}
\text{Add F6,F8,F2} & \text{yes} & \text{Add (Load 2)} & 0 & \text{Add1} & \text{Load2} \\
\text{Div F10, F0, F6} & \text{yes} & \text{Div (Load 1)} & & 0 & 0 \\
\text{Sub F8, F6, F2} & \text{yes} & \text{Sub (Load1)} & 0 & \text{Load2} & \text{Qj} \Rightarrow 0 \text{ means Vj has a value} \\
\text{Mul F0, F2, F4} & \text{yes} & \text{Mul (Load 2)} & & 0 & 0 \\
\text{Load F2, 45(r3)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\text{Load F6, 34(r2)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\end{array} \]

- The tag (or name) can be:
  - Zero (or a special pattern) meaning that the value is indeed a value
  - The name of a load buffer
  - The name of a reservation station within a functional unit

- A reservation station consists of:
  - The operation to be performed
  - 2 pairs (value, tag) \( (V_j, Q_j), (V_k, Q_k) \)
  - A flag indicating whether the accompanying f-u is busy or not

\[ \text{Instruction} \quad \text{Issue} \quad \text{Execute} \quad \text{Write result} \]

\[ \begin{array}{cccc}
\text{Add F6,F8,F2} & \text{yes} & \text{Add (Load 2)} & 0 & \text{Add1} & \text{Load2} \\
\text{Div F10, F0, F6} & \text{yes} & \text{Div (Load 1)} & & 0 & 0 \\
\text{Sub F8, F6, F2} & \text{yes} & \text{Sub (Load1)} & 0 & \text{Load2} & \text{Qj} \Rightarrow 0 \text{ means Vj has a value} \\
\text{Mul F0, F2, F4} & \text{yes} & \text{Mul (Load 2)} & & 0 & 0 \\
\text{Load F2, 45(r3)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\text{Load F6, 34(r2)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\end{array} \]

- The tag (or name) can be:
  - Zero (or a special pattern) meaning that the value is indeed a value
  - The name of a load buffer
  - The name of a reservation station within a functional unit

- A reservation station consists of:
  - The operation to be performed
  - 2 pairs (value, tag) \( (V_j, Q_j), (V_k, Q_k) \)
  - A flag indicating whether the accompanying f-u is busy or not

\[ \text{Instruction} \quad \text{Issue} \quad \text{Execute} \quad \text{Write result} \]

\[ \begin{array}{cccc}
\text{Add F6,F8,F2} & \text{yes} & \text{Add (Load 2)} & 0 & \text{Add1} & \text{Load2} \\
\text{Div F10, F0, F6} & \text{yes} & \text{Div (Load 1)} & & 0 & 0 \\
\text{Sub F8, F6, F2} & \text{yes} & \text{Sub (Load1)} & 0 & \text{Load2} & \text{Qj} \Rightarrow 0 \text{ means Vj has a value} \\
\text{Mul F0, F2, F4} & \text{yes} & \text{Mul (Load 2)} & & 0 & 0 \\
\text{Load F2, 45(r3)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\text{Load F6, 34(r2)} & \text{yes} & \text{Load 2} & & 0 & 0 \\
\end{array} \]
Other checks/possibilities

- In the example in the book there is no load/store dependencies but since they can happen
  - Load/store buffers must keep the addresses of the operands
  - On load, check if there is a corresponding address in store buffers. If so, get the value/tag from there (load/store buffers have tags)
  - Better yet, have load/store functional units (still needs checking)
- The Tomasulo engine was intended only for f-p operations. We need to generalize to include
  - Handling branches, exceptions etc
  - In-order completion
  - More general register renaming mechanisms
  - Multiple instruction issues