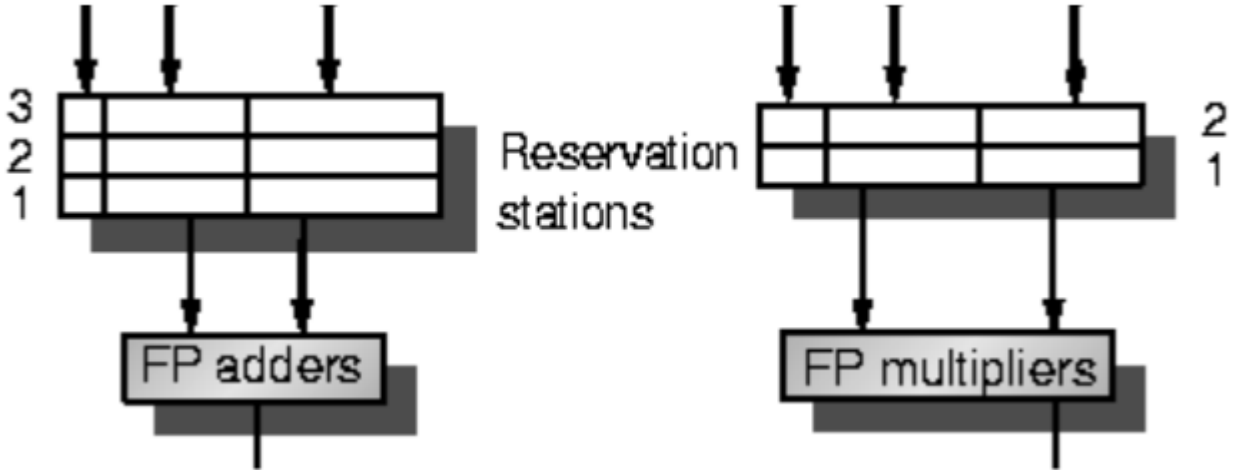
09.Tomasulo Algorithm

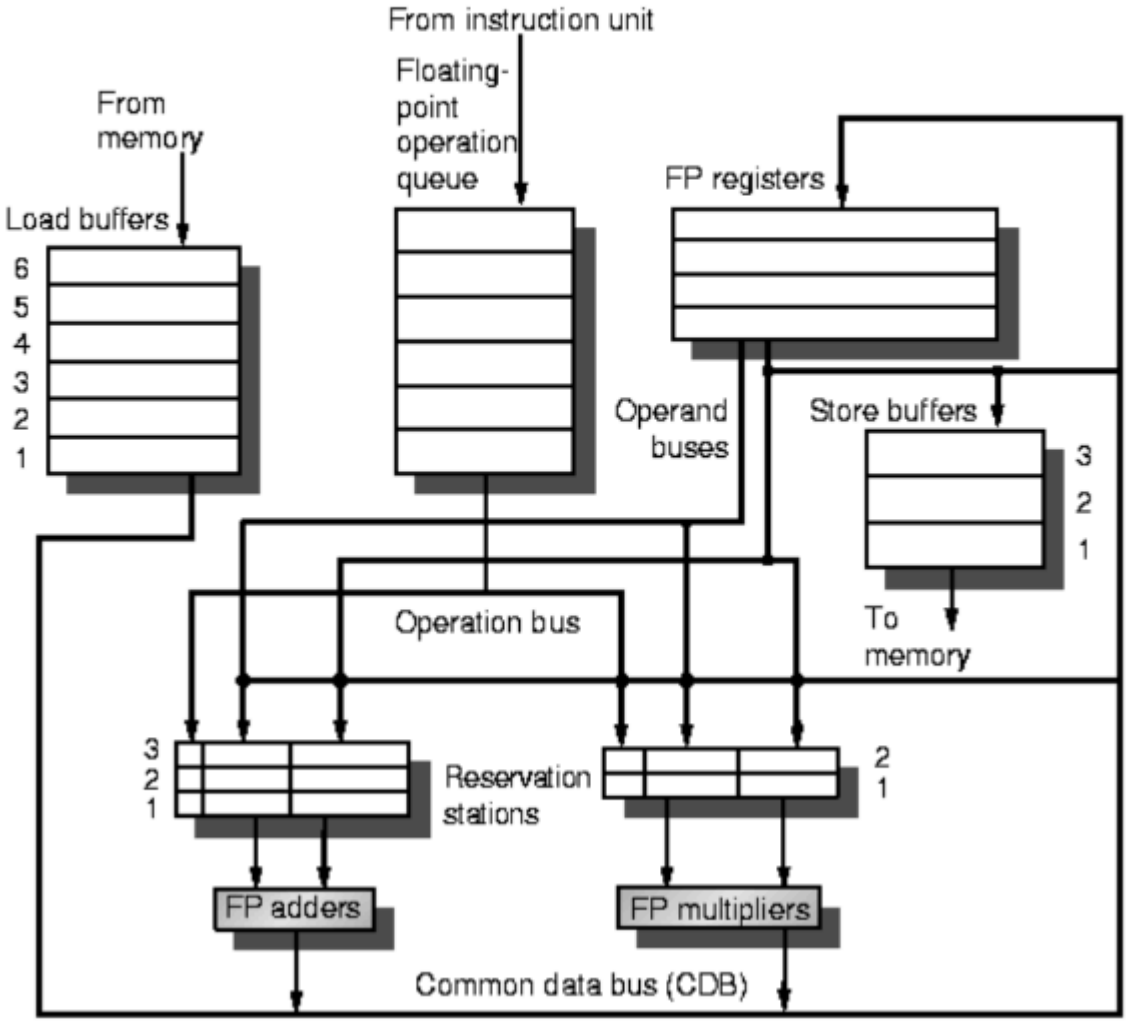
Another dynamic algorithm: allows execution to proceed in the presence of dependencies
The idea is to schedule instruction as soon as possible, not having special compilers for optimisation.
The idea is that the control logic and the buffers are distributed with FUs (vs. centralized in scoreboard). Operand buffers are called Reservation Stations. Each instruction is an entry of a reservation station. Its operands are replaced by values or pointers (Register Renaming).



Register Renaming allows to:

- Avoid WAR and WAW hazards(The value is copied in the reservation station of the FU so don't care about changing in the register as the right value is in the reservation station)
- Reservation stations can be consider as additional register but they are more than registers (so can do better optimizations than a compiler).
  Results are dispatched to other FUs through a Common Data Bus (data+source).
  Load/Stores treated as FUs.

Tomasulo Algorithm for an FPU



Reservation Station Components

- Tag identifying the RS
- OP = the operation to perform on the component(Similar to the one in the scoreboard).
- Vj, Vk = Value of the source operands
- Qj,Qk = Pointers to RS that produce Vj,Vk
  Zero value = Source op. is already available in Vj or Vk, if Q-pointer is different to zero the information is not available/the operation is already terminated
- Busy = Indicates RS Busy
  Note: Only one of V-field or Q-field is valid for each operand(V and Q fields are mutually exclusive, cannot be different from zero at the same time)
- RF and the Store buffer have a Value (V) and a Pointer (Q) field.Pointer (Q) field corresponds to number of reservation station producing the result to be stored in RF or store buffer. If zero $\Rightarrow$ no active instructions producing the result(RF or store buffer content is the correct value).
- Load buffers have an address field (A), and a busy field.
- Store Buffers have also an address field (A).

- A: To hold info for memory address calculation for load/store. Initially contains the instruction offset (immediate field); after address calculation stores the effective address.

3-stage of the Tomasulo Algorithm: ISSUE

Get an instruction I from the queue.

- If it is an FP op. Check if a RS is empty (i.e., check for structural hazards)
  Rename registers
  WAR resolution
- If I writes Rx, read by an instruction K already issued, K knows already the value of Rx or knows what instructio will write it. So the RF can be linked to I
  WAW resolution
- Since we use in-order issue, the RF can be linked to I
  We get an instruction from the queue. See if the reservation station is empty(check for structural hazards)and see if we can procede. Solve WaR. When we issue an operation there are two cases: the value is ready in the register file or you know where is the value and copy it in the Reservation Station

3-stages of the Tomasulo Algorithm: EXECUTION

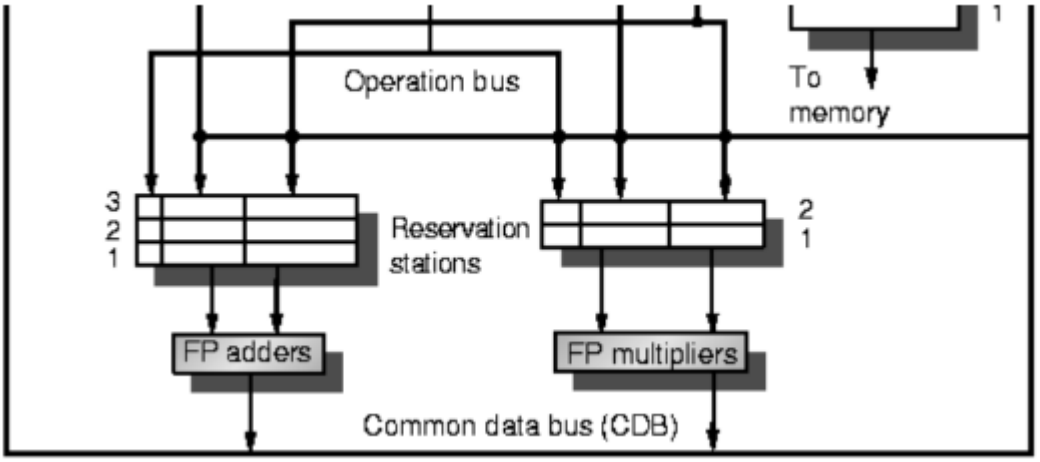
When both operands are ready then execute.
If not ready, watch the common data bus for results.

- By delaying execution until operands are available, RAW hazards are avoided. Notice that several instructions could become ready in the same clock cycle for the same FU.
  Load and stores: two-step process
- First step: compute effective address, place it in load or store buffer.
- Loads in Load Buffer execute as soon as memory unit is available
- Stores in store buffer wait for the value to be stored before being sent to memory unit
  Different from the Scoreboard as the value are copied inside the Reservation Station and so don't cause Hazards(No need of notification for data availability). If the value are ready you can exit, if not you monitor the common data bus to see when the operand not ready becomes available. Automatically RaW hazards because you delay the execution of an operation until everything is ready. You could have more than one operation that become ready at the same clock cycle so when value are ready you can start multiple operation at the same time. Load and Store are treated as FUs as you have a busy not busy flag and they are using the value in the Reservation Station.
  You wait

3-stage of the Tomasulo Algorithm: WRITE

When result is available: write on Common Data Bus and from there into RF and into all RSs (including store buffers) waiting for this result

Stores write data to memory during this stage
Mark reservation stations available.



Focus on load/store

Loads and stores go through a functional unit for effective address computation before proceeding to effective load and store buffers.
Loads take a second execution step to access memory, then go to Write Result to send the value from memory to RF and/or RS;
Stores complete their execution in their Write Result stage (writes data to memory)
All writes occur in Write Result – simplifying Tomasulo algorithm.
A Load and a Store can be done in different order, provided they access different memory locations. That could be problematic if they are using the same address as the order of execution is important and can change the result(can be done in parallel but needs to maintains the order of execution).
Otherwise, a WAR (interchange load-store sequence) or a RAW (interchange store-load sequence) may result (WAW if two stores are interchanged). Loads can be reordered freely.
To detect such hazards: data memory addresses associated with any earlier memory operation must have been computed by the CPU (e.g.: address computation executed in program order)
Load executed out of order with previous store: assume address computed in program order. When Load address has been computed, it can be compared with A fields in active Store buffers: in the case of a match, Load is not sent to Load buffer until conflicting store completes.
Stores must check for matching addresses in both Load and Store buffers (dynamic disambiguation, alternative to static disambiguation performed by the compiler)
Drawback: amount of hardware required.
Each RS must contain a fast associative buffer; single CDB may limit performance.

Tomasulo versus Scoreboard

Tomasulo(IBM)	Scoreboard(CDC)
Issue window size=14	Issue window size=5

Tomasulo(IBM)	Scoreboard(CDC)
No issue on structural hazards	No issue on structural hazards
WAR, WAW avoided with renaming	Stall the completion for WAW and WAR hazards
Broadcast results from FU	Results written back on registers.
Control distributed on RS	Control centralized through the Scoreboard
Allows loop unrolling in HW	

Control & buffers distributed with Function Units (FU) vs. centralized in scoreboard;
FU buffers called “reservation stations”; have pending operands
Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming:

- avoids WAR, WAW hazards
- More reservation stations than registers, so can do optimizations compilers can’t
  Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
  Load and Stores treated as FUs with RSs as well
  Integer instructions can go past branches, allowing FP ops beyond basic block in FP queue
  Tomasulo can compute faster than the Scoreboard.