Hardware-Based Speculation

Exploiting More ILP

- Branch prediction reduces stalls but may not be sufficient to generate the desired amount of ILP
- One way to overcome control dependencies is with **speculation**
  - Make a guess and execute program as if our guess is correct
  - Need mechanisms to handle the case when the speculation is incorrect
- Can do some speculation in the compiler
  - We saw this previously with reordered / duplicated instructions around branches

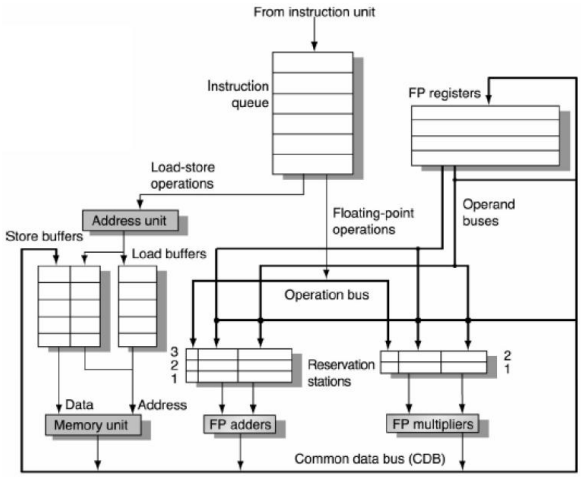
Hardware-Based Speculation

- Extends the idea of dynamic scheduling with three key ideas:
  1. Dynamic branch prediction
  2. Speculation to allow the execution of instructions before control dependencies are resolved
  3. Dynamic scheduling to deal with scheduling different combinations of basic blocks
     - What we saw earlier was within a basic block
- Modern processors started using speculation around the introduction of the PowerPC 603, Intel Pentium II and extend Tomasulo's approach to support speculation

Speculating with Tomasulo

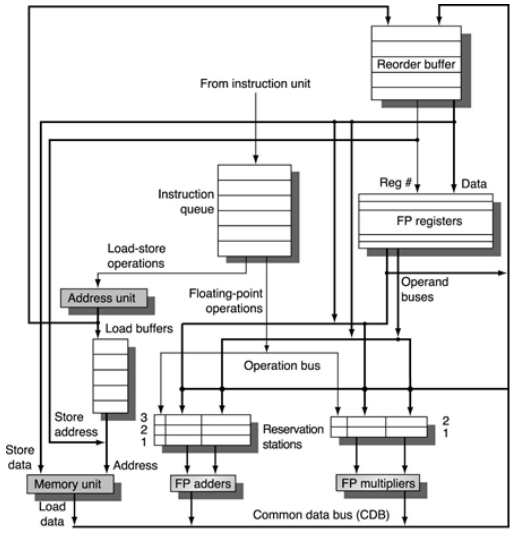
- Separate execution from completion
  - Allow instructions to execute speculatively but do not let instructions update registers or memory until they are no longer speculative
- Instruction Commit
  - After an instruction is no longer speculative it is allowed to make register and memory updates
- Allow instructions to execute and complete out of order but force them to commit in order
- Add a hardware buffer, called the **reorder buffer** (**ROB**), with registers to hold the result of an instruction between completion and commit
  - Acts as a FIFO queue in order issued

Original Tomasulo Architecture



Tomasulo and Reorder Buffer

- Sits between Execution and Register File
- Source of operands
- In this case integrated with Store buffer
- Reservation stations use ROB slot as a tag
- Instructions commit at head of ROB FIFO queue
  - Easy to undo speculated instructions on mispredicted branches or on exceptions



ROB Data Structure

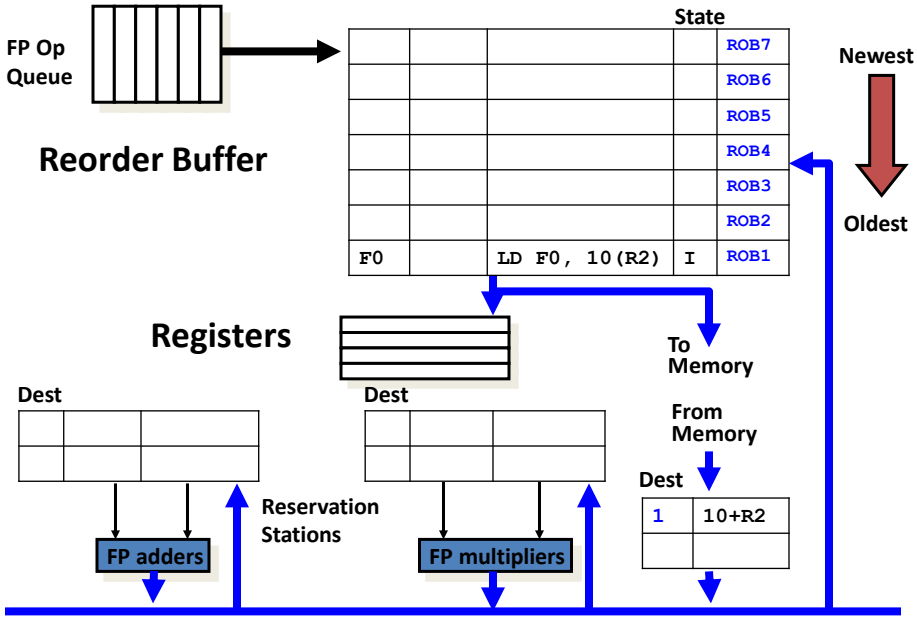
- Instruction Type Field
  - Indicates whether the instruction is a branch, store, or register operation
- Destination Field
  - Register number for loads, ALU ops, or memory address for stores
- Value Field
  - Holds the value of the instruction result until instruction commits
- Ready Field
  - Indicates if instruction has completed execution and the value is ready

Instruction Execution

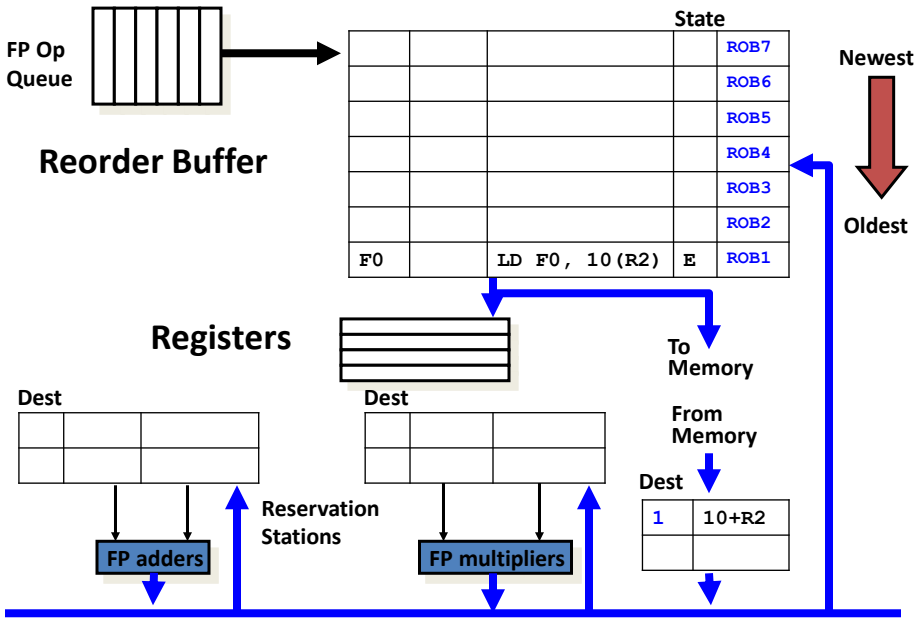
1. Issue: Get an instruction from the Instruction Queue
   - If the reservation station and the ROB has a free slot (no structural hazard), issue the instruction to the reservation station and the ROB, send operands to the reservation station if available in the register file or the ROB. The allocated ROB slot number is sent to the reservation station to use as a tag when placing data on the CDB.
2. Execution: Operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result
3. Write result: Finish execution (WB)
   - Write on CDB to all awaiting units and to the ROB using the tag; mark reservation station available
4. Commit: Update register or memory with the ROB result
   - When an instruction reaches the head of the ROB and results are present, update the register with the result or store to memory and remove the instruction from the ROB
   - **If an incorrectly predicted branch reaches the head of the ROB, flush the ROB, and restart at the correct successor of the branch**

Blue text = Change from Tomasulo

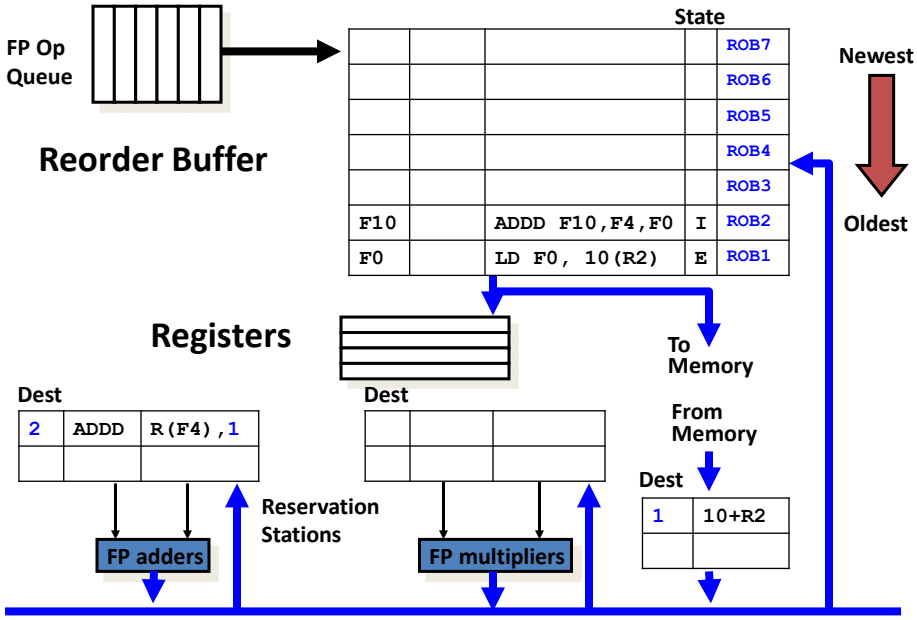
Tomasulo With ROB



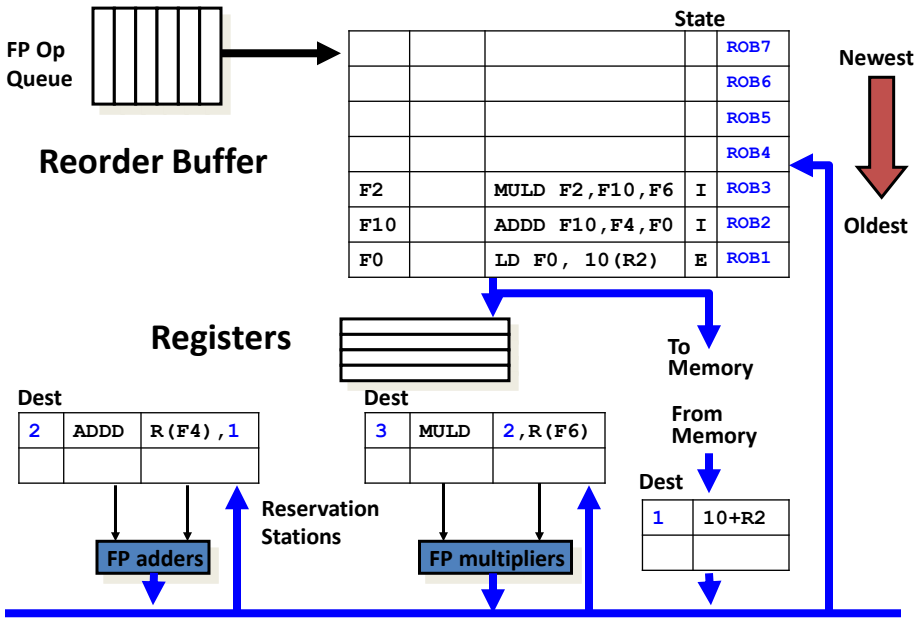
Tomasulo With ROB



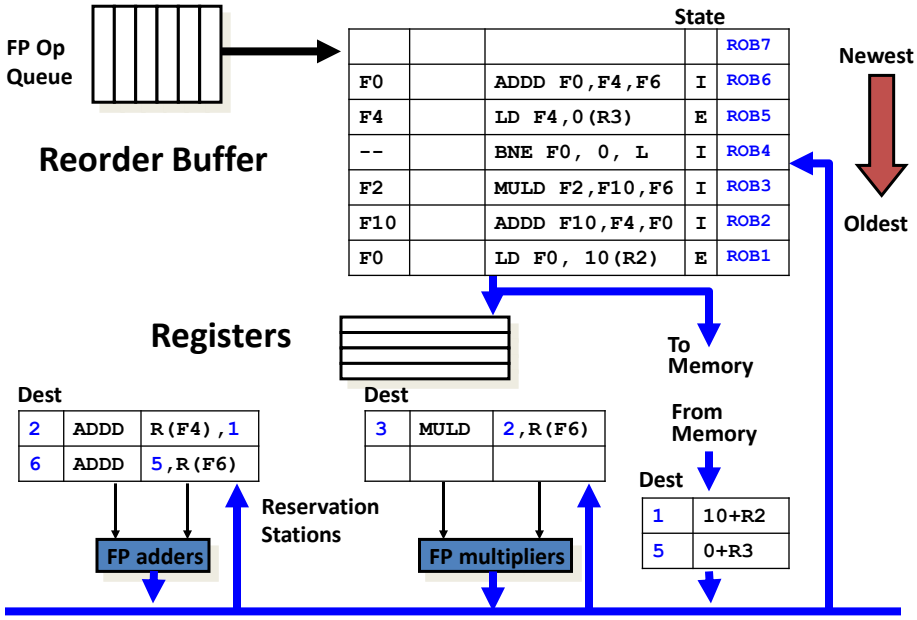
Tomasulo With ROB



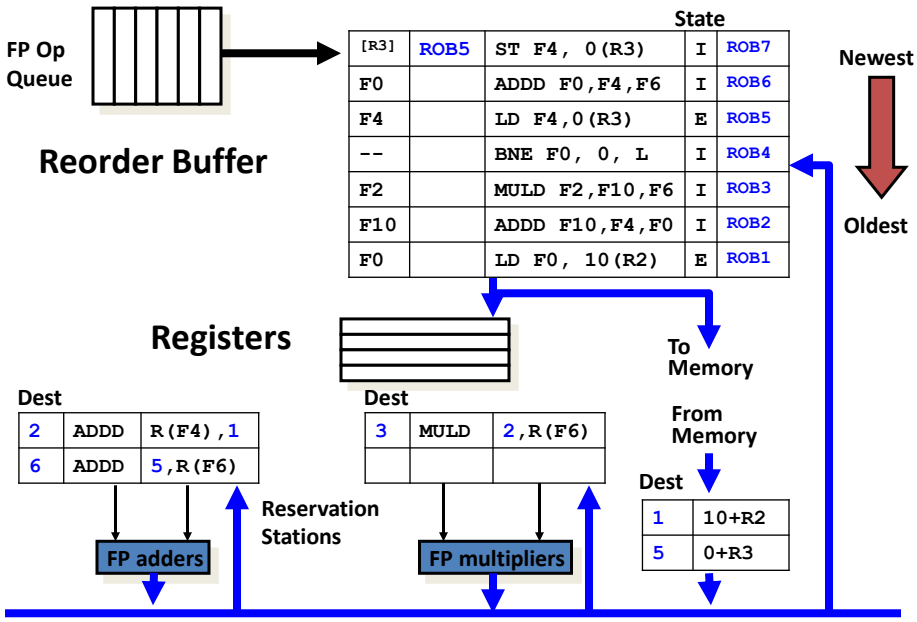
Tomasulo With ROB



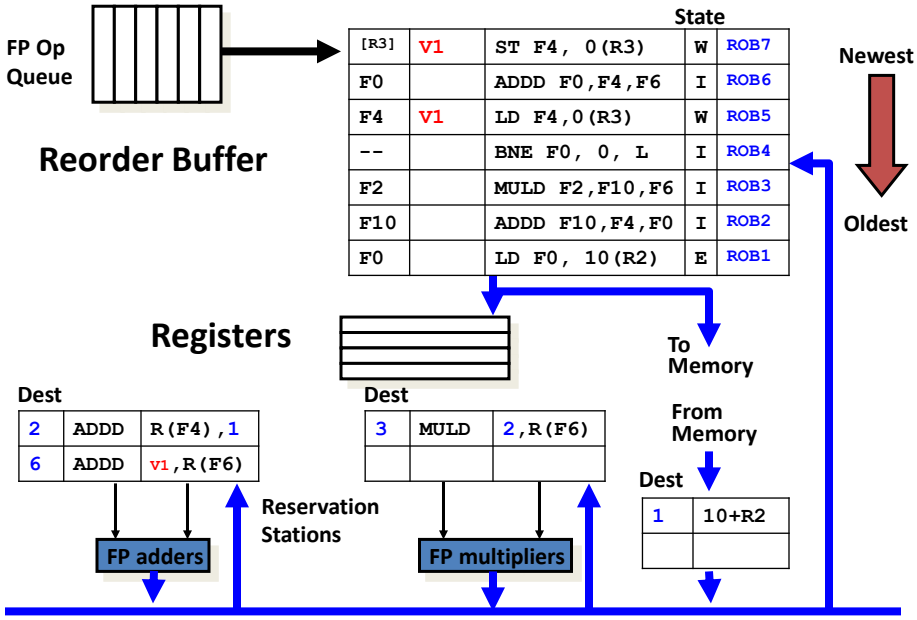
Tomasulo With ROB



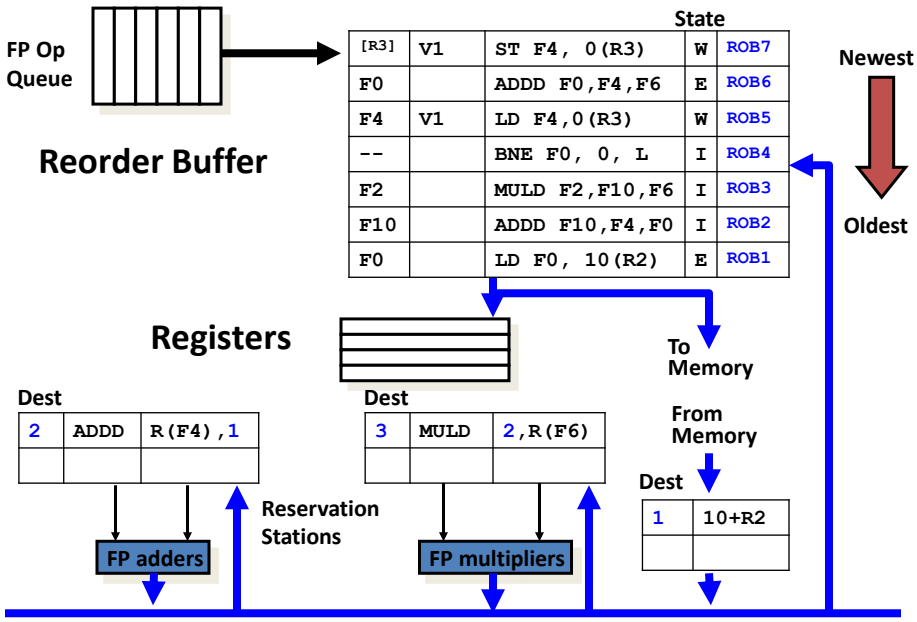
Tomasulo With ROB



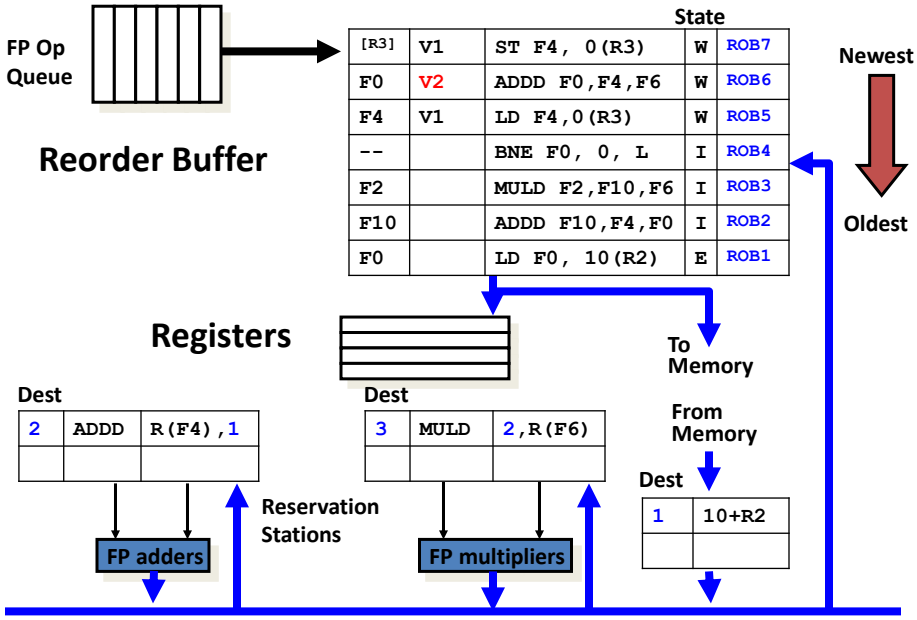
Tomasulo With ROB



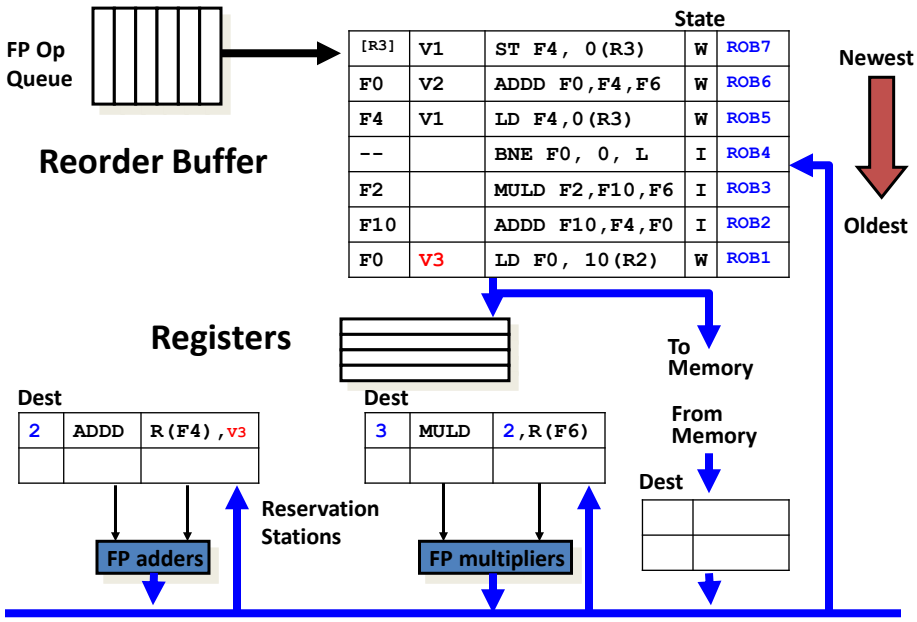
Tomasulo With ROB



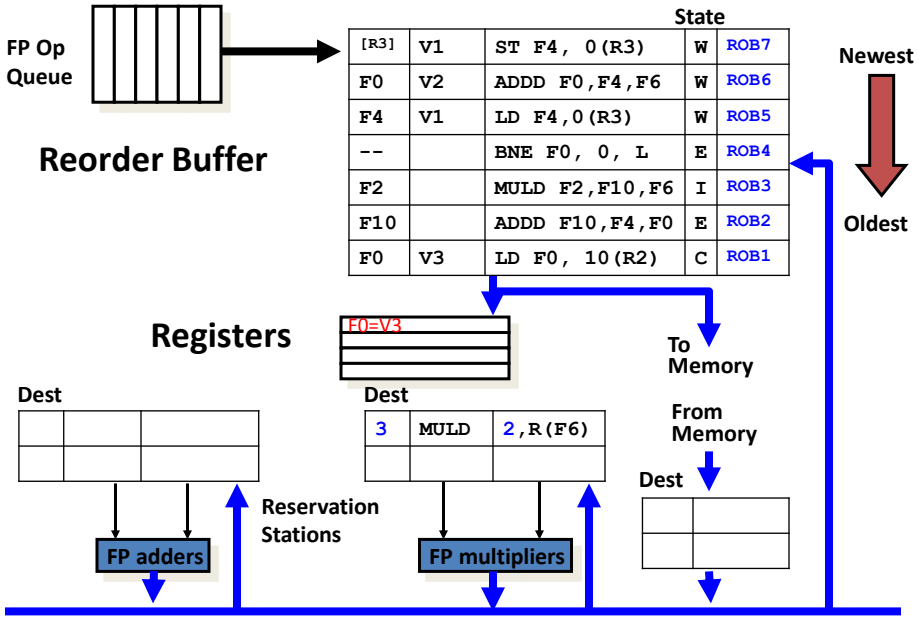
Tomasulo With ROB



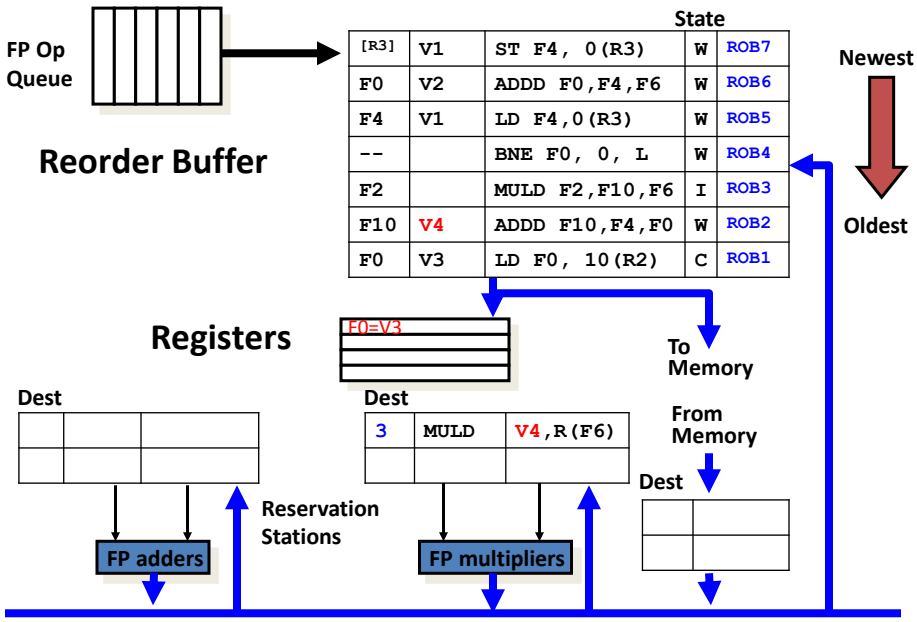
Tomasulo With ROB



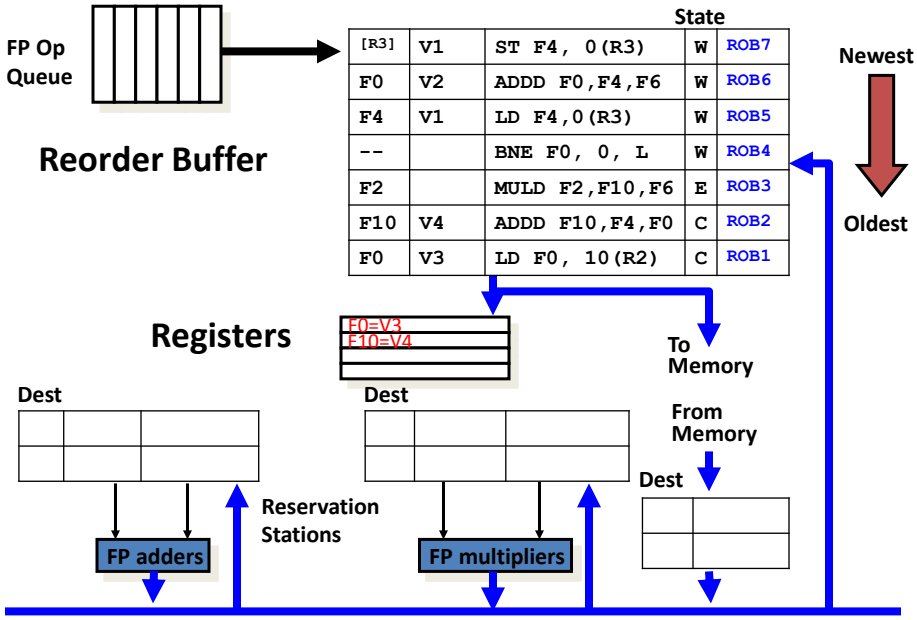
Tomasulo With ROB



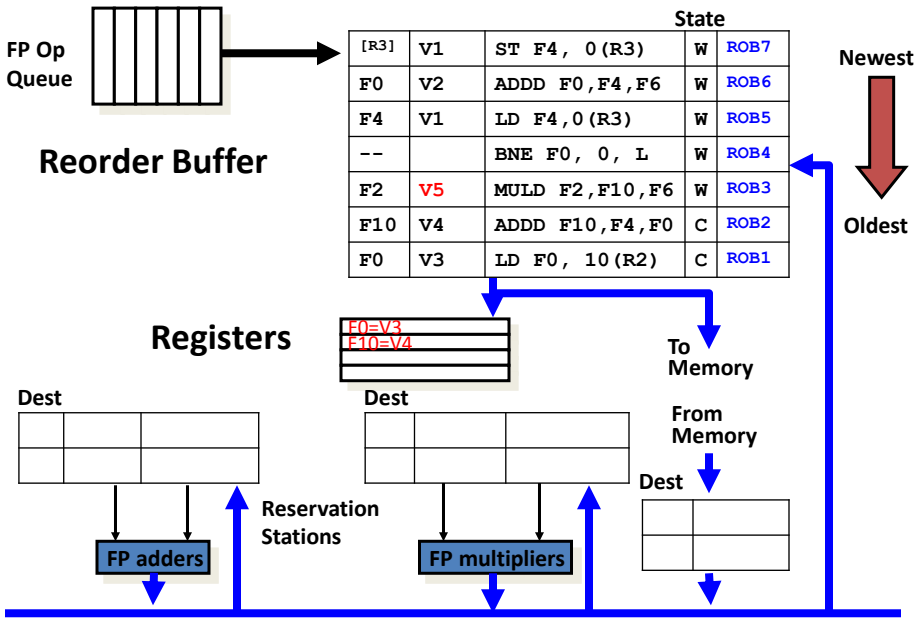
Tomasulo With ROB



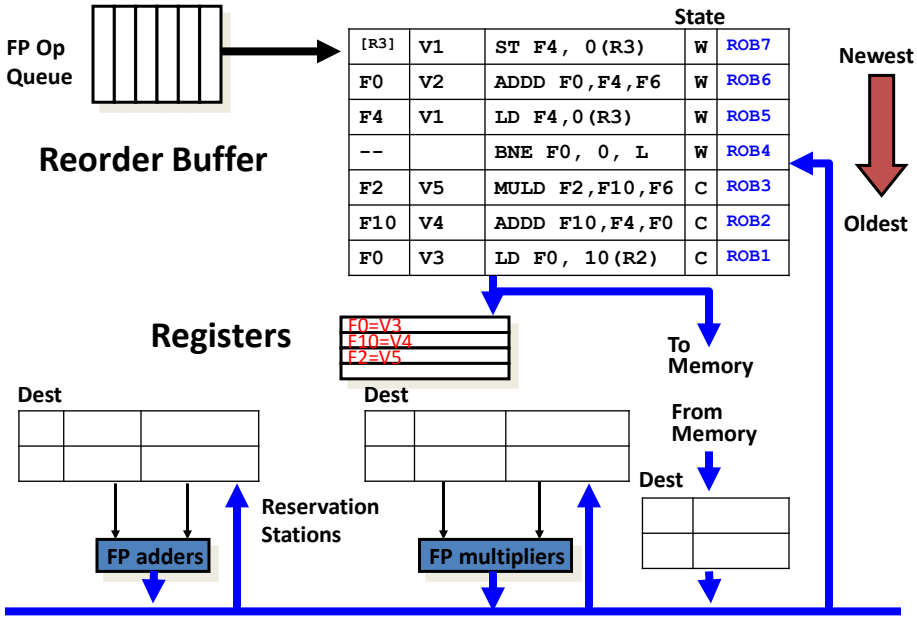
Tomasulo With ROB



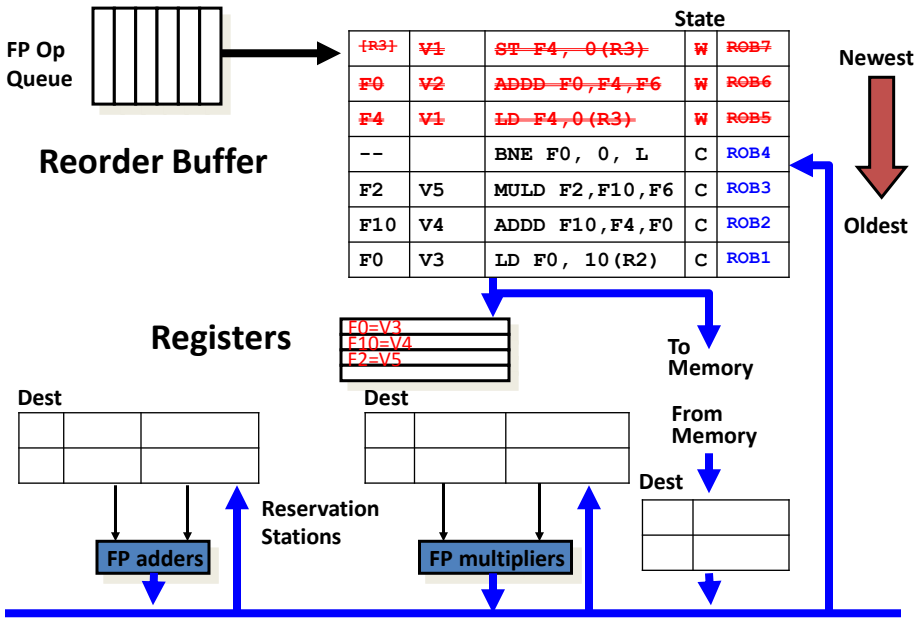
Tomasulo With ROB



Tomasulo With ROB



Tomasulo With ROB



Avoiding Memory Hazards

- A store only updates memory when it reaches the head of the ROB
  - Otherwise WAW and WAR hazards are possible
  - By waiting to reach the head memory is updated in order and no earlier loads or stores can still be pending
- If a load accesses a memory location written to by an earlier store then it cannot perform the memory access until the store has written the data
  - Prevents RAW hazard through memory

Reorder Buffer Implementation

- In practice
  - Try to recover as early as possible after a branch is mispredicted rather than wait until branch reaches the head
  - Performance in speculative processors more sensitive to branch prediction
    - Higher cost of misprediction
- Exceptions
  - Don't recognize the exception until it is ready to commit
  - Could try to handle exceptions as they arise and earlier branches resolved, but more challenging