Advanced Computer Architectures

(High Performance Processors and Systems)

Hardware Based Speculation

Politecnico di Milano

v1

Alessandro Verosimile <alessandro.verosimile@polimi.it>

Marco D. Santambrogio <marco.santambrogio@polimi.it>

EXE ON APRIL 9
LIVE @25.S.2

Outline

- HW-based speculation main idea
- Reorder Buffer
- An example

HW-based Speculation

- HW-based Speculation combines 3 ideas:
  - Dynamic Branch Prediction to choose which instruction to execute
  - Dynamic Scheduling supporting out-of-order execution but in-order commit to prevent any irrevocable actions (such as register update or taking exception) until an instruction commits
  - Speculation to execute instructions before control dependences are resolved

HW-based Speculation

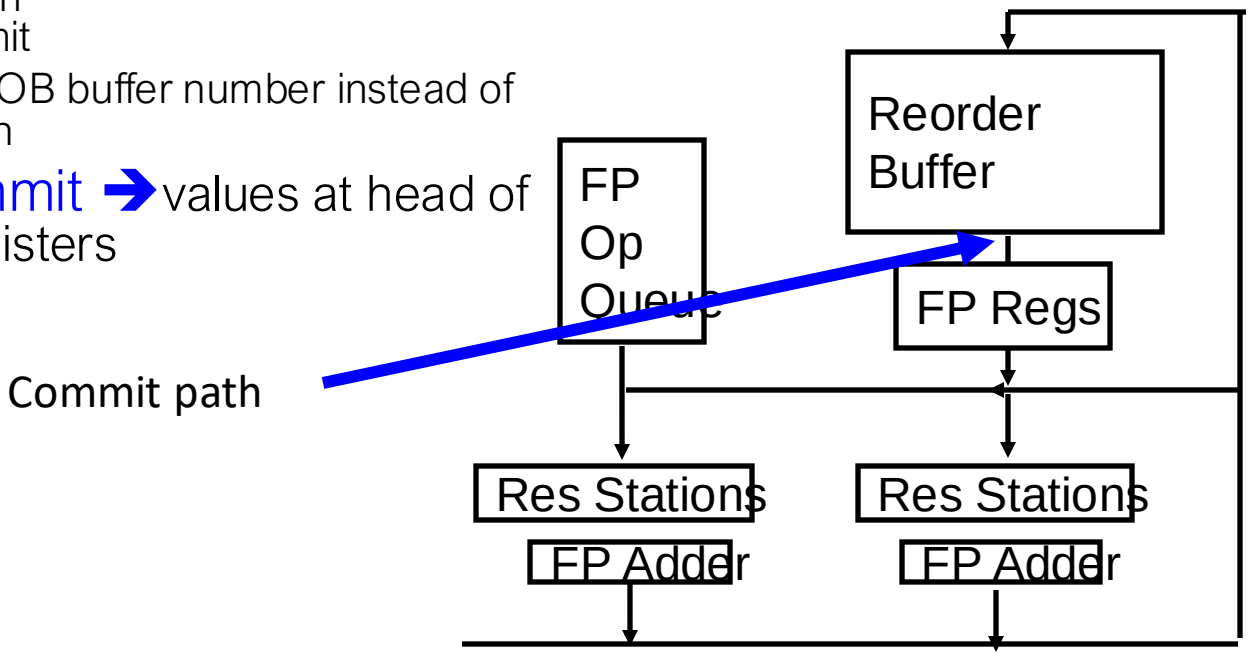
- The key idea behind speculation is:
  - to issue and execute instructions dependent on a branch before the branch outcome is known;
  - to allow instructions to execute out-of-order but to force them to commit in-order;
  - to prevent any irrevocable action (such as updating state or taking an exception) until an instruction commits;
- When an instruction is no longer speculative, we allow it to update the register file or memory (instruction commit).

HW-based Speculation

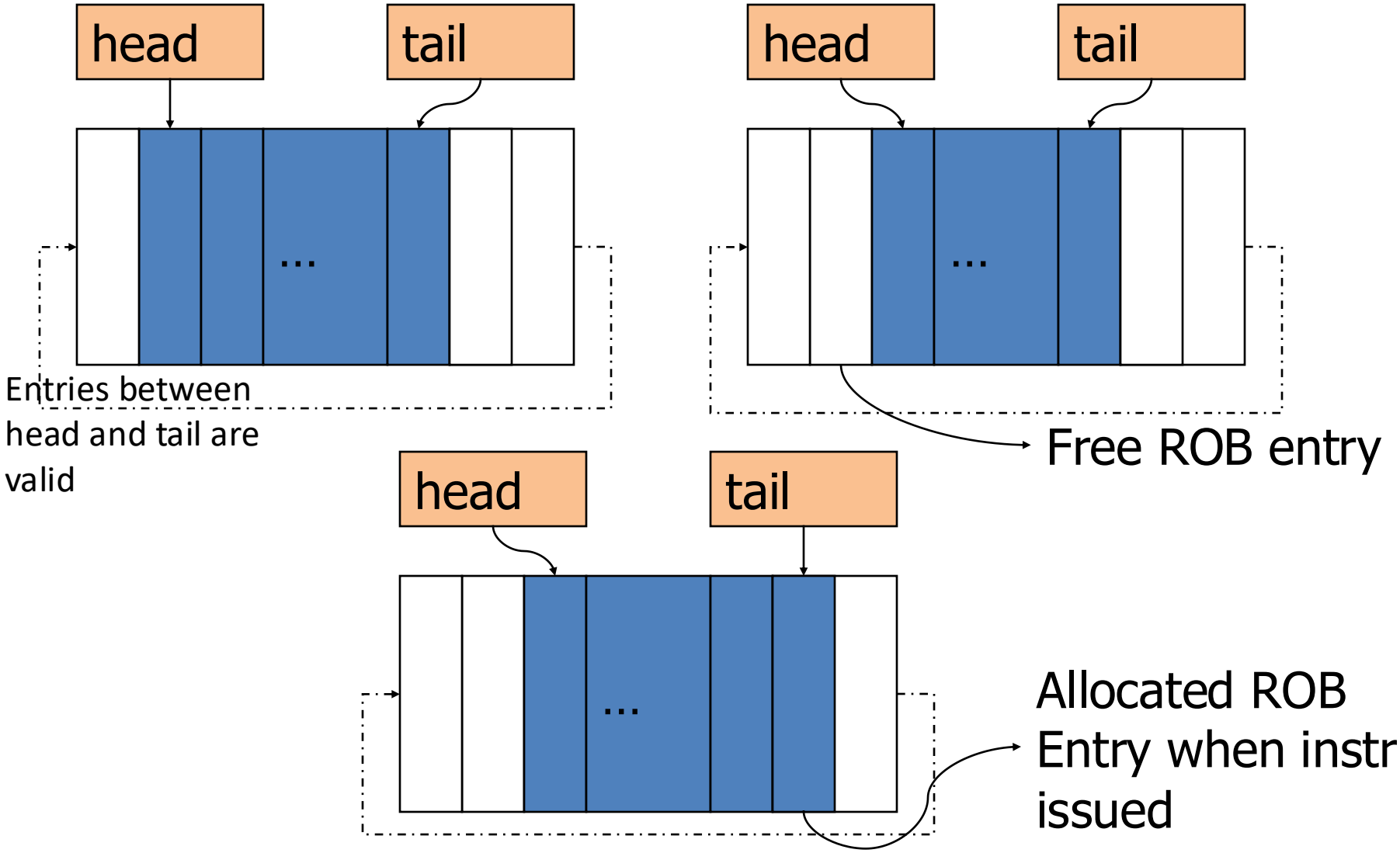
- Outcome of branches is speculated and program is executed as if speculation was correct (without speculation, the simple dynamic scheduling would only fetch and decode, not execute!)
- Mechanisms are necessary to handle incorrect speculation – hardware speculation extends dynamic scheduling beyond a branch (i.e. behind the basic block)

Reorder Buffer

- Holds instructions in FIFO order, exactly as issued
- When instructions complete, results placed into ROB
  - Supplies operands to other instruction between execution complete & commit
  - Tag results with ROB buffer number instead of reservation station
- Instructions commit ➔ values at head of ROB placed in registers



ROB: Circular Buffer with Head/Tail Pointers



Separating Completion from Commit

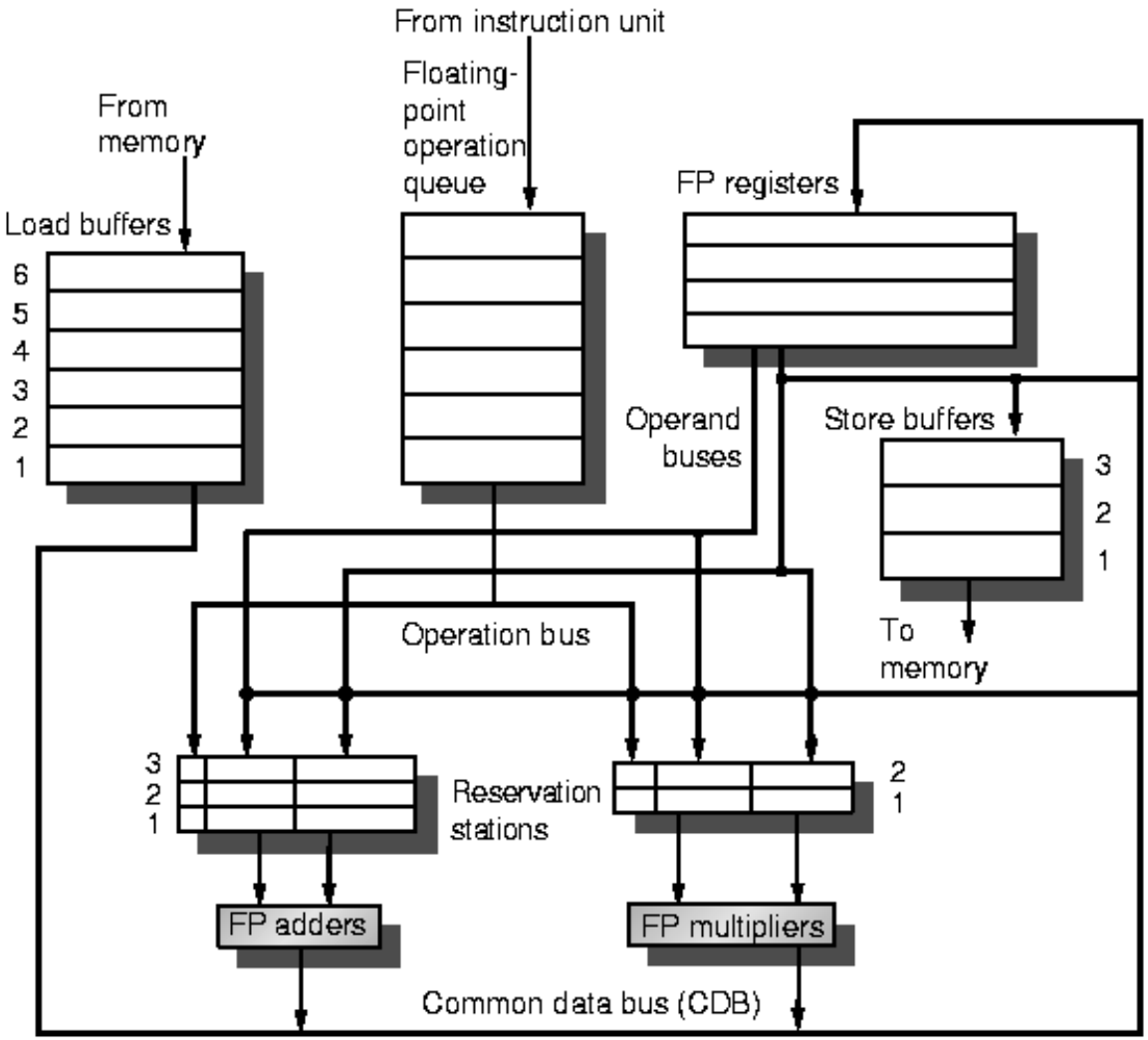
- Re-order buffer holds register results from completion until commit
  - Entries allocated in program order during decode
  - Buffers completed values and exception state until in-order commit point
  - Completed values can be used by dependents before committed (bypassing)
  - Each entry holds program counter, instruction type, destination register specifier and value if any, and exception status (info often compressed to save hardware)
- Memory reordering needs special data structures
  - Speculative store address and data buffers
  - Speculative load address and data buffers

Relationship between precise interrupts and speculation

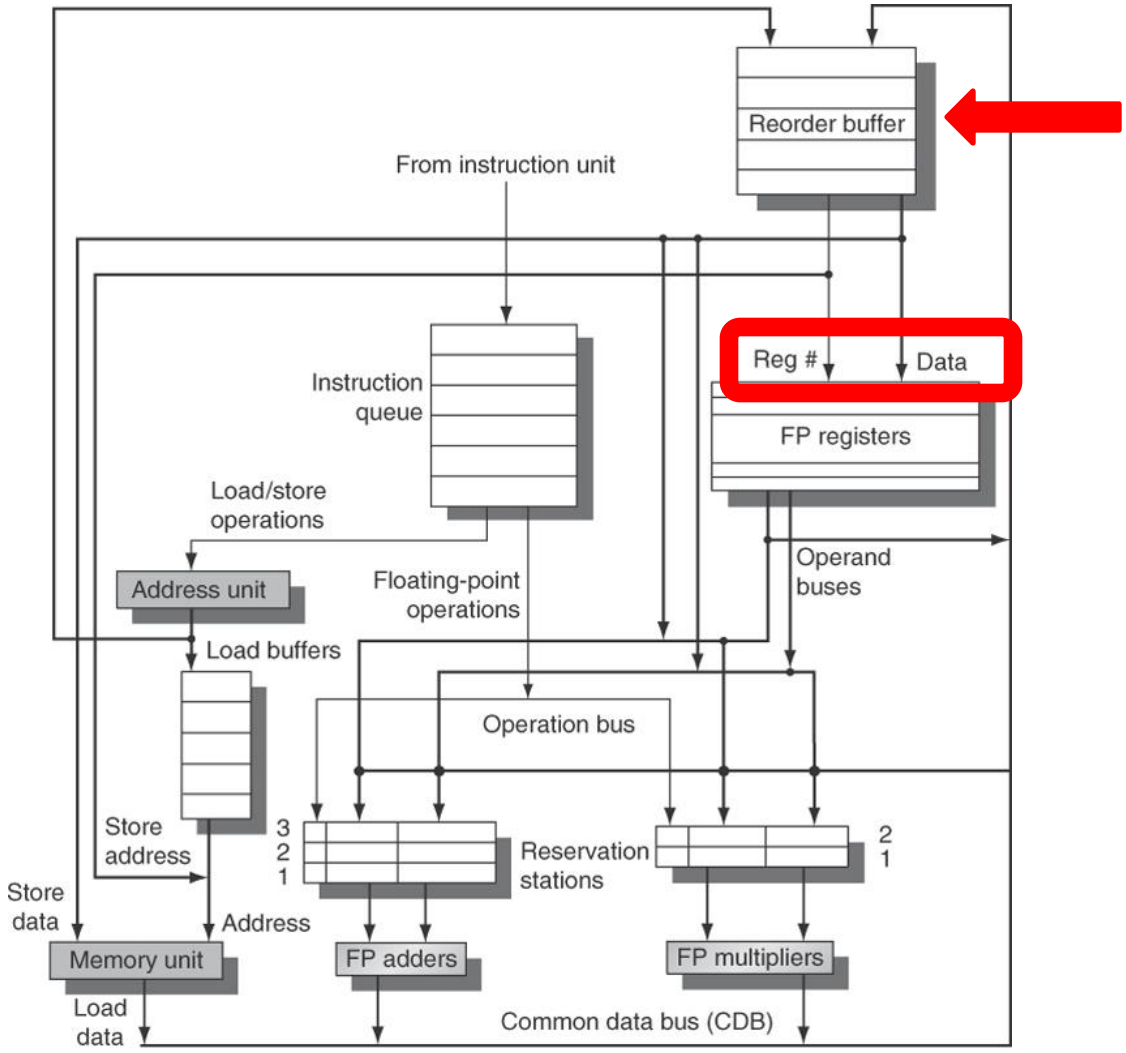
- Speculation: guess and check
- Important for branch prediction:
  - Need to "take our best shot" at predicting branch direction
- If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly:
  - This is exactly the same as precise exceptions!
- Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*

MORE ON EXCEPTIONS/INTERUPTS @ VIDEO C9

Tomasulo Algorithm for an FPU



Speculative Tomasulo Architecture for an FPU



Four Steps of Speculative Tomasulo Algorithm 1/2

1. Issue: get instruction from FP Op Queue
   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. Execution: operate on operands (EX)
   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. Write result: finish execution (WB)
   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. Commit: update register with reorder result
   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

Steps of Speculative Tomasulo's Algorithm 2/2

4. Commit: 3 different possible sequences:
   1. Normal commit: instruction reaches the head of the ROB, result is present in the buffer. Result is stored in the register, instruction is removed from ROB;
   2. Store commit: as above, but memory rather than register is updated;
   3. Instruction is a branch with incorrect prediction: it indicates that speculation was wrong. ROB is flushed ("graduation") , execution restarts at correct successor of the branch.
      If the branch was correctly predicted, branch is finished

Speculative Tomasulo's Algorithm

- Tomasulo's "Boosting" needs a buffer for uncommitted results (reorder buffer).

Each entry in ROB contains four fields:

- Instruction type field – indicates whether instruction is a branch (no destination result), a store (has memory address destination), or a load/ALU (register destination)
- Destination field: supplies register number (for loads and ALU instructions) or memory address (for stores) where results should be written;
- Value field (used to hold value of result until instruction commits)
- Ready field: indicates that instruction has completed execution, value is ready

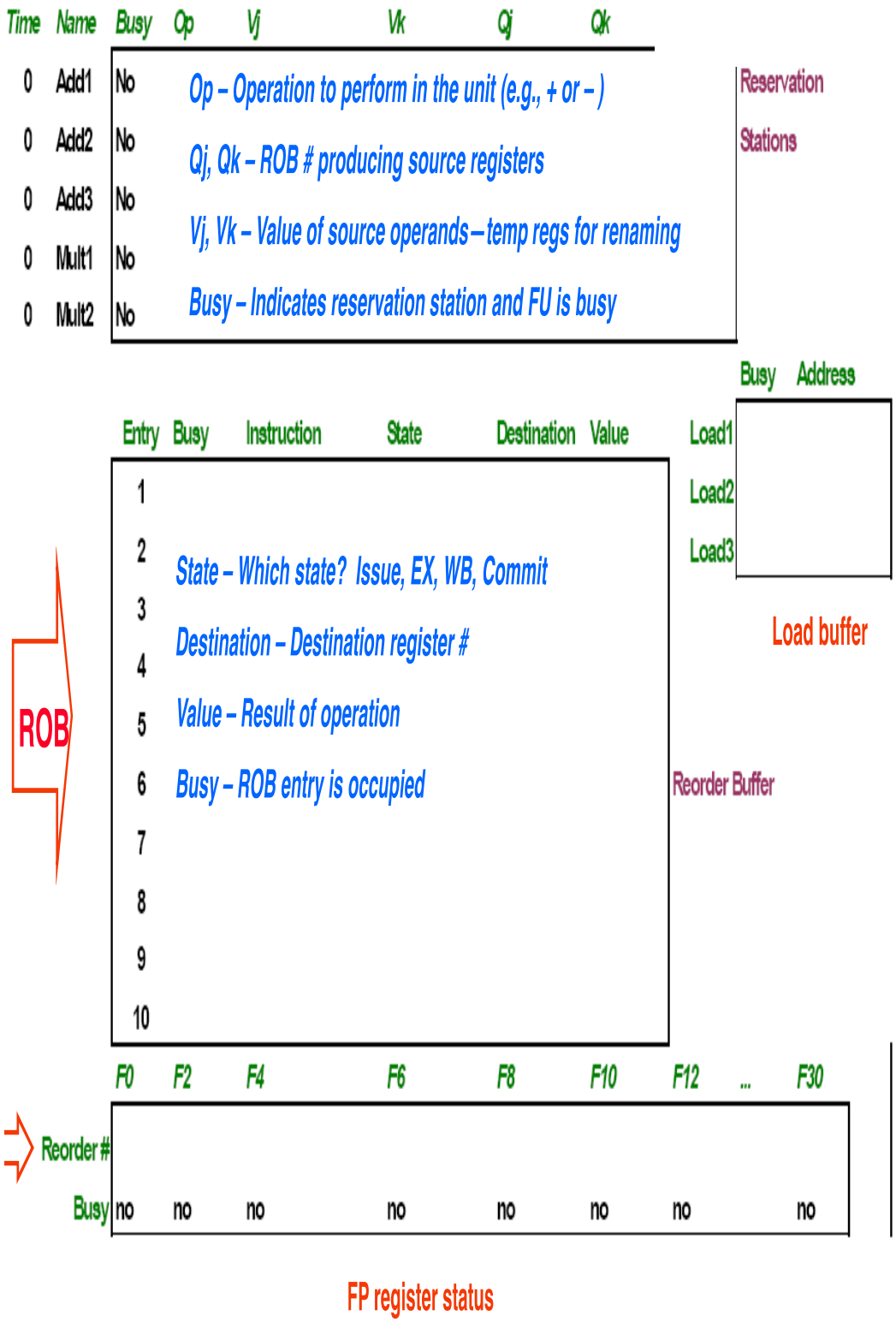
ReOrder Buffer Extension

- ROB completely replaces store buffers: stores execute in two steps, the second one when instruction commits;
- Renaming function of reservation stations completely replaced by ROB;
- Reservation stations now only queue operations (and operands) to FUs between the time they issue and the time they begin execution;
- Results are tagged with ROB entry number rather than with RS number ⇨ ROB entry assigned to instruction must be tracked in the reservation stations.

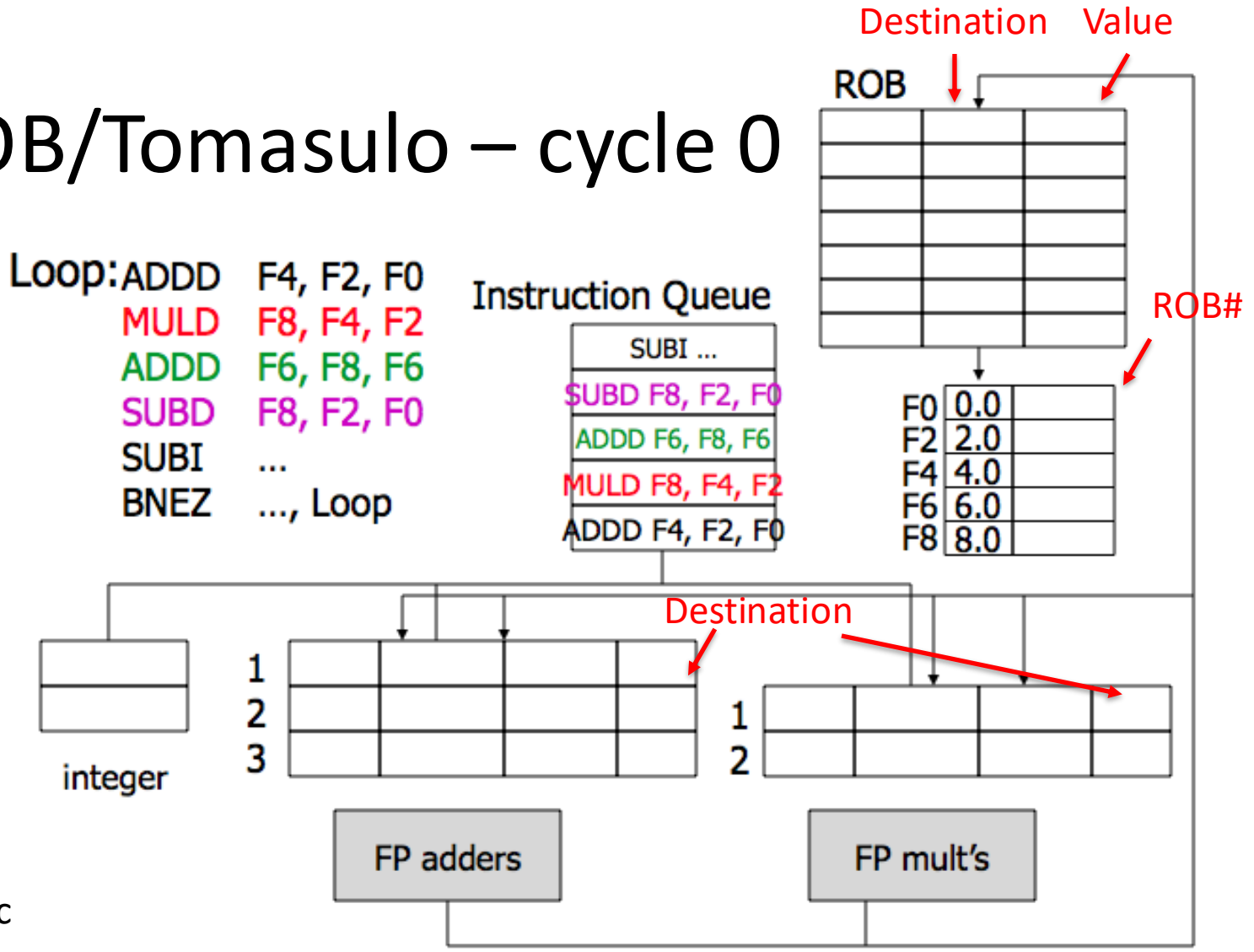
ReOrder Buffer Extension

- All instructions excluding incorrectly predicted branches (or incorrectly speculated loads) commit when reaching head of ROB;
- Incorrectly predicted branch reaches head of ROB ⇨ wrong speculation is indicated, ROB is flushed, execution restarts at correct successor of branch.
- Speculative actions are easily undone.
- Processors with ROB can dynamically execute while maintaining a precise interrupt model:
  - if instruction I_j causes interrupt, CPU waits until I_j reaches the head of the ROB and takes the interrupt, flushing all other pending instructions.

Tomasulo with ROB



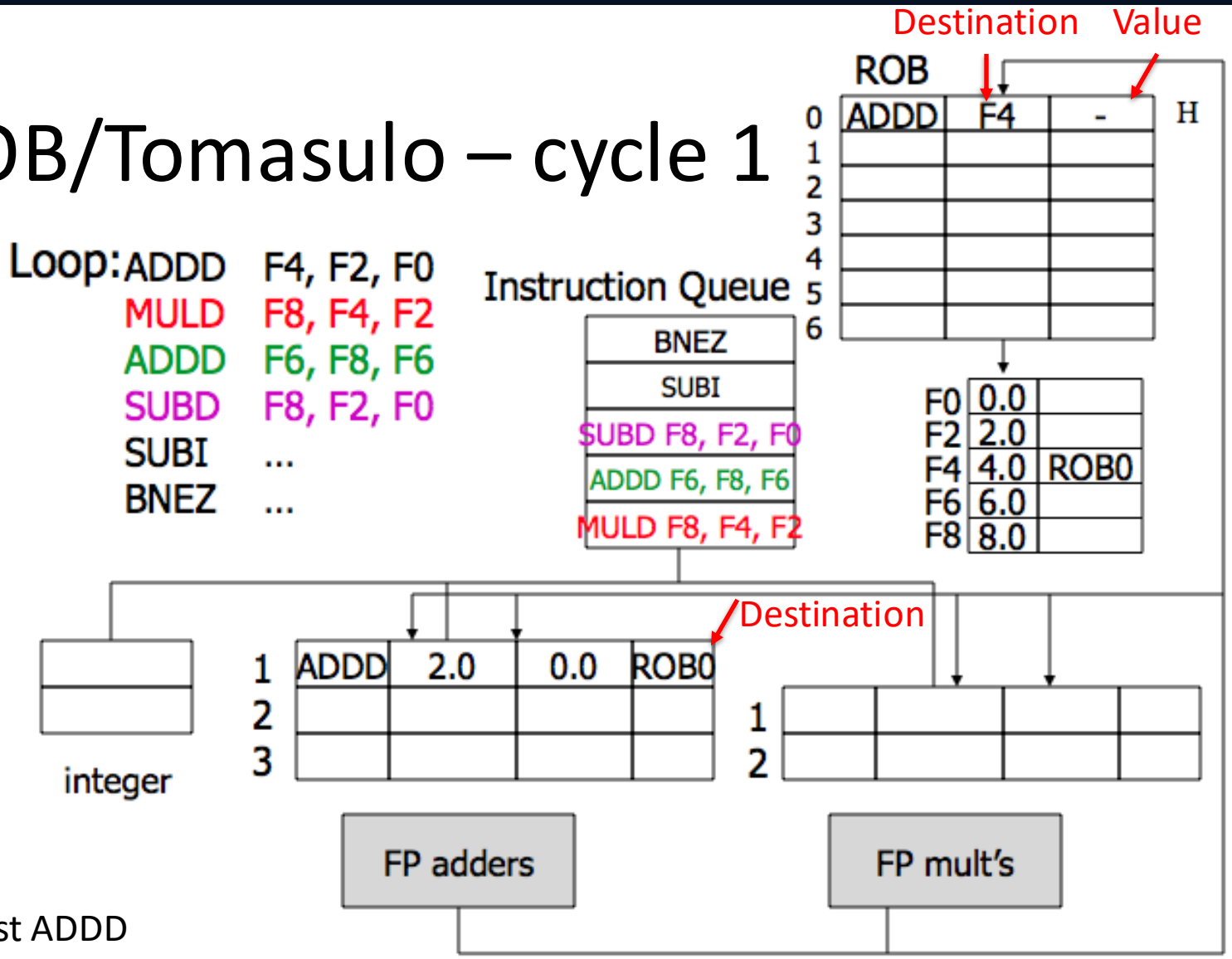
ROB/Tomasulo – cycle 0



ADDD 3cc
MULT 10cc

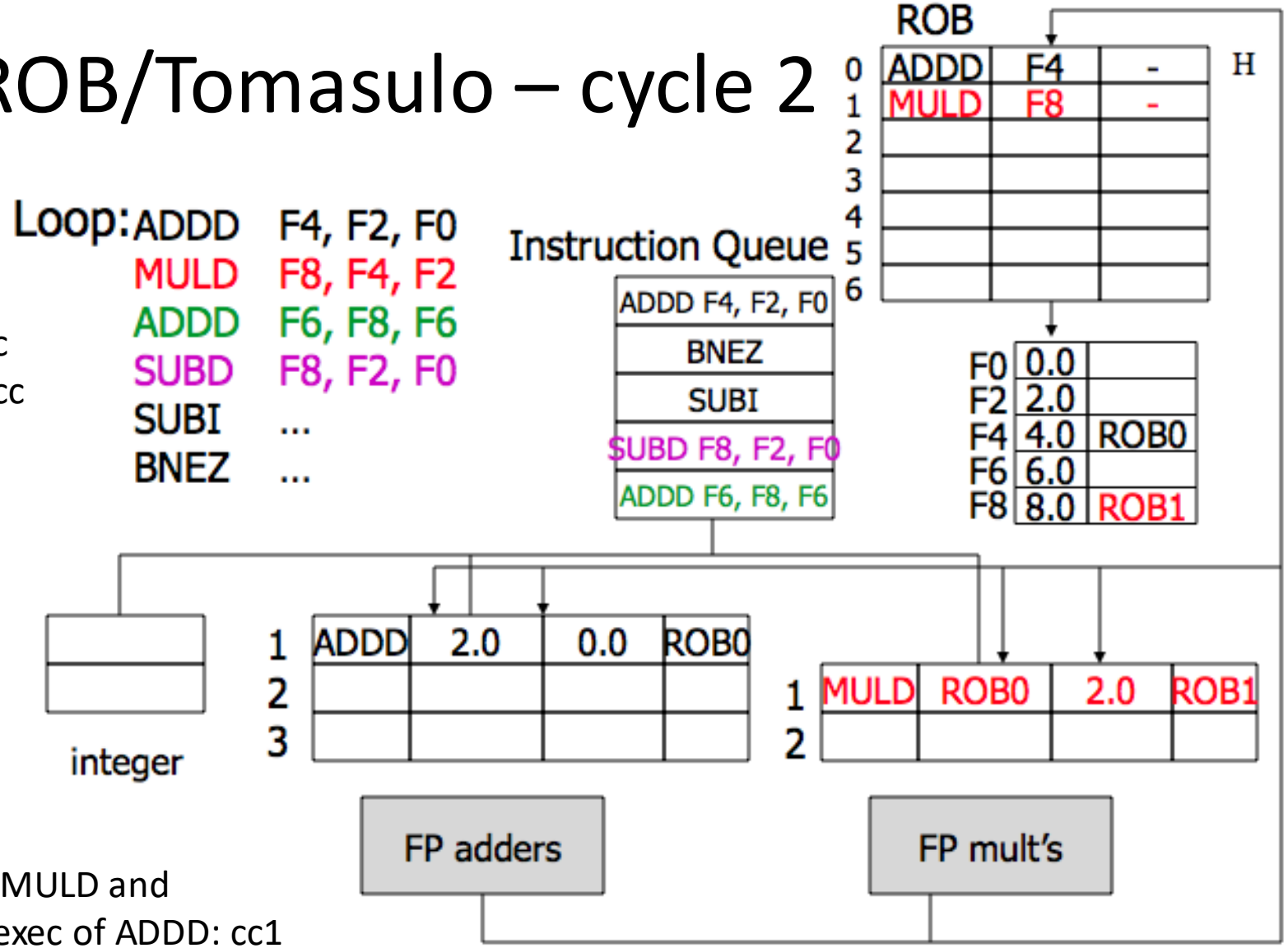
ROB/Tomasulo – cycle 1

ADDD 3cc
MULT 10cc



Issue first ADDD

ROB/Tomasulo – cycle 2

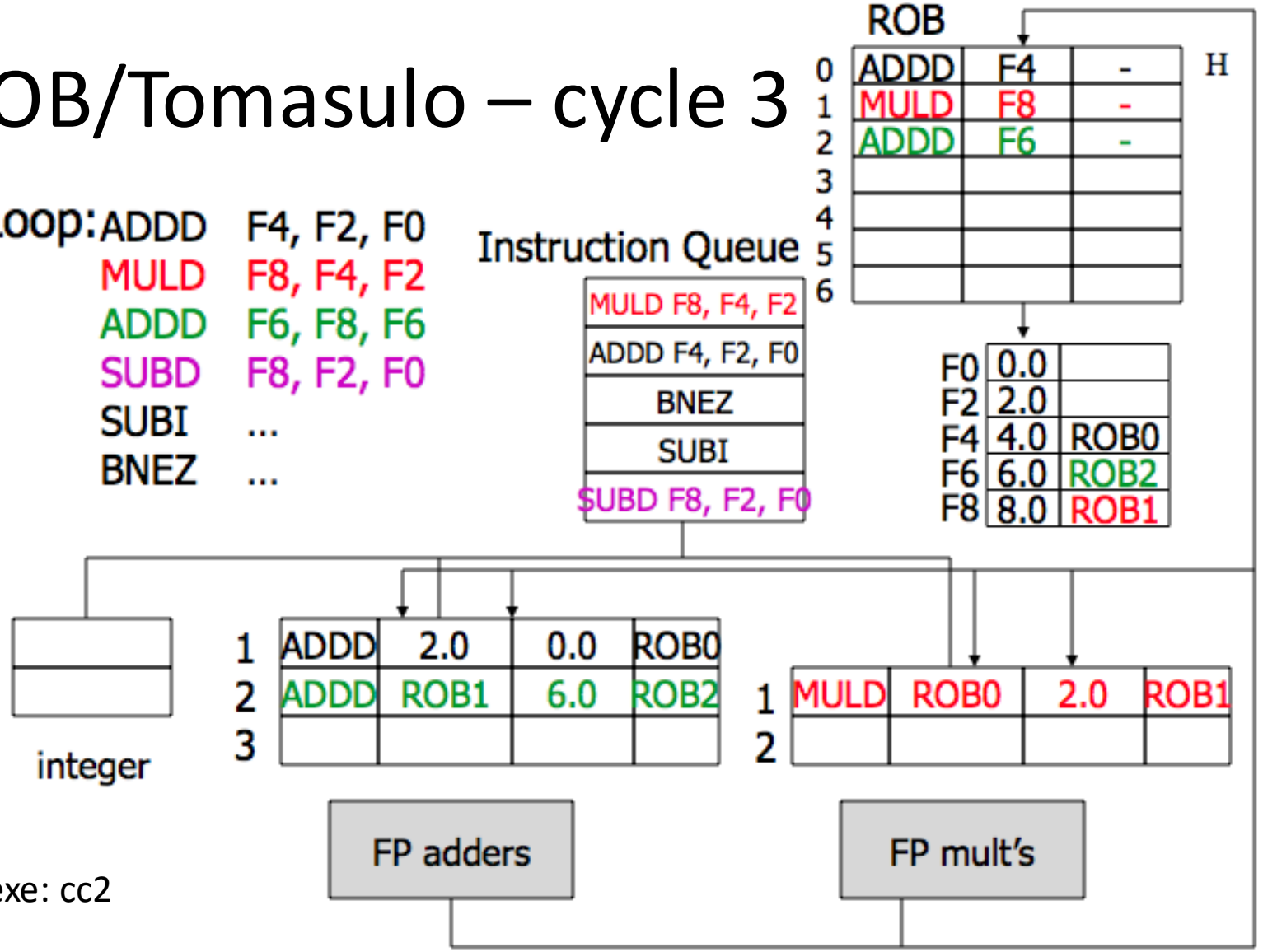


Issue MULD and
start exec of ADDD: cc1

ROB/Tomasulo – cycle 3

ADDD 3cc
MULT 10cc

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```

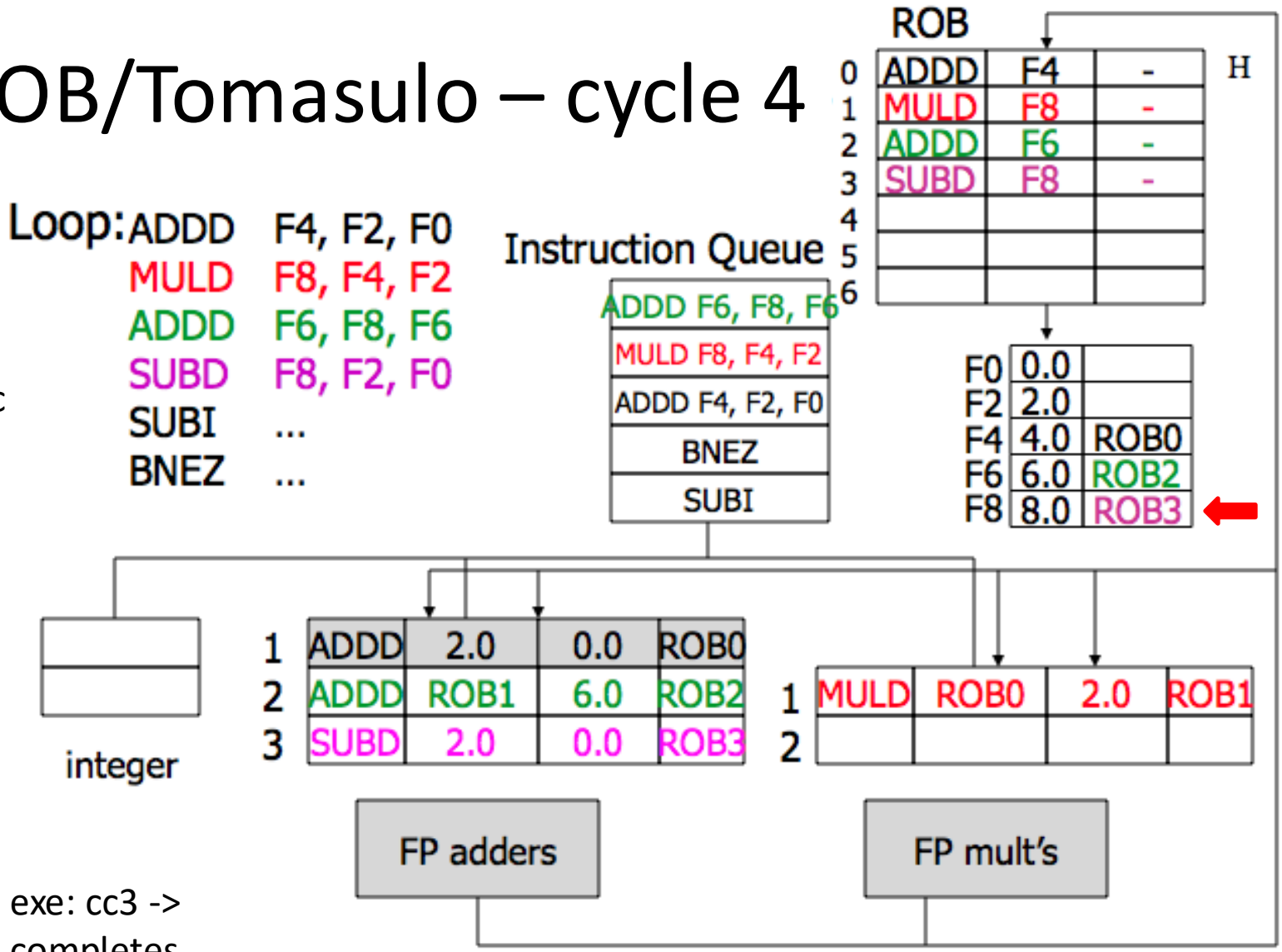


ADDD exe: cc2

ROB/Tomasulo – cycle 4

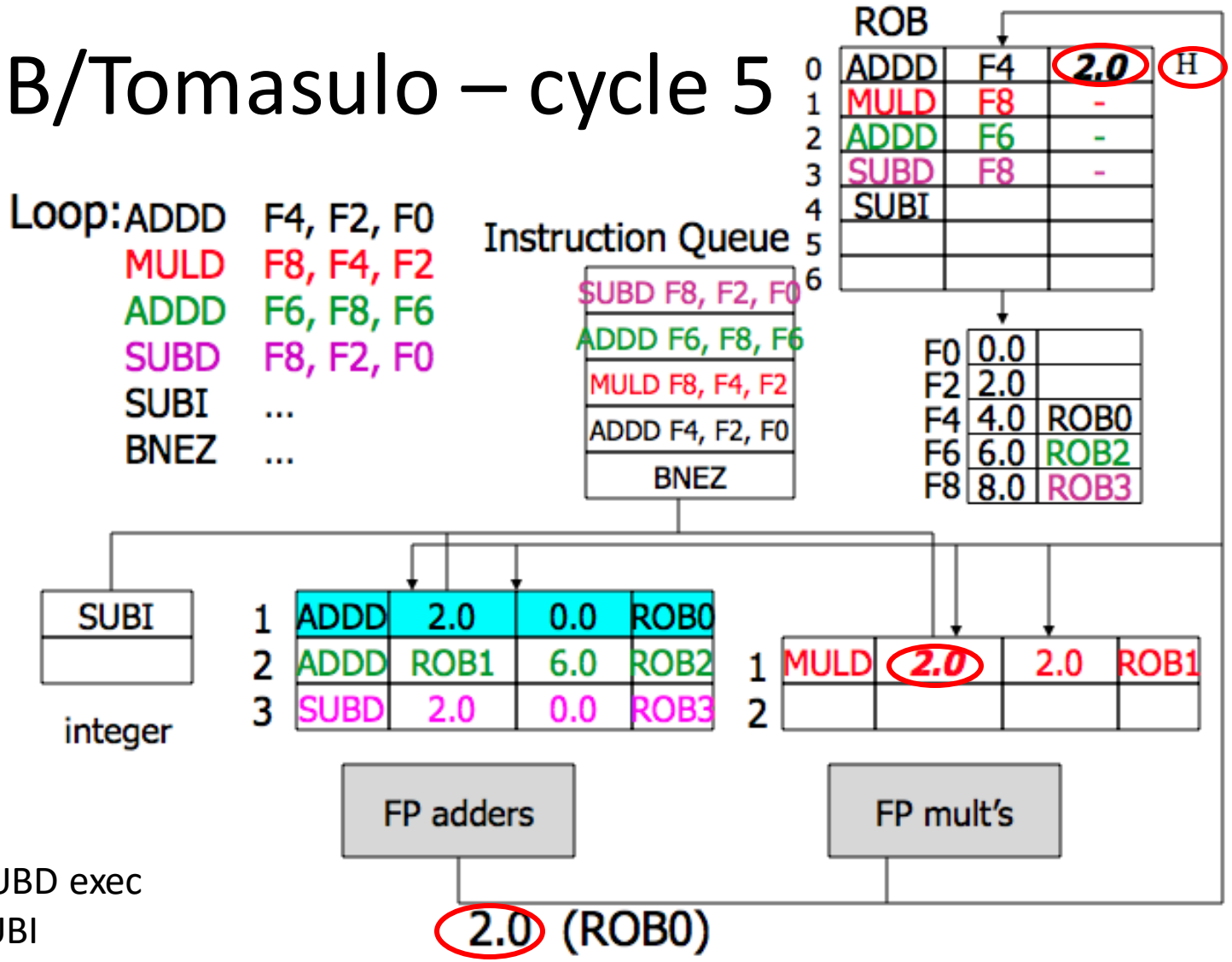
ADDD 3cc
MULT 10cc

```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



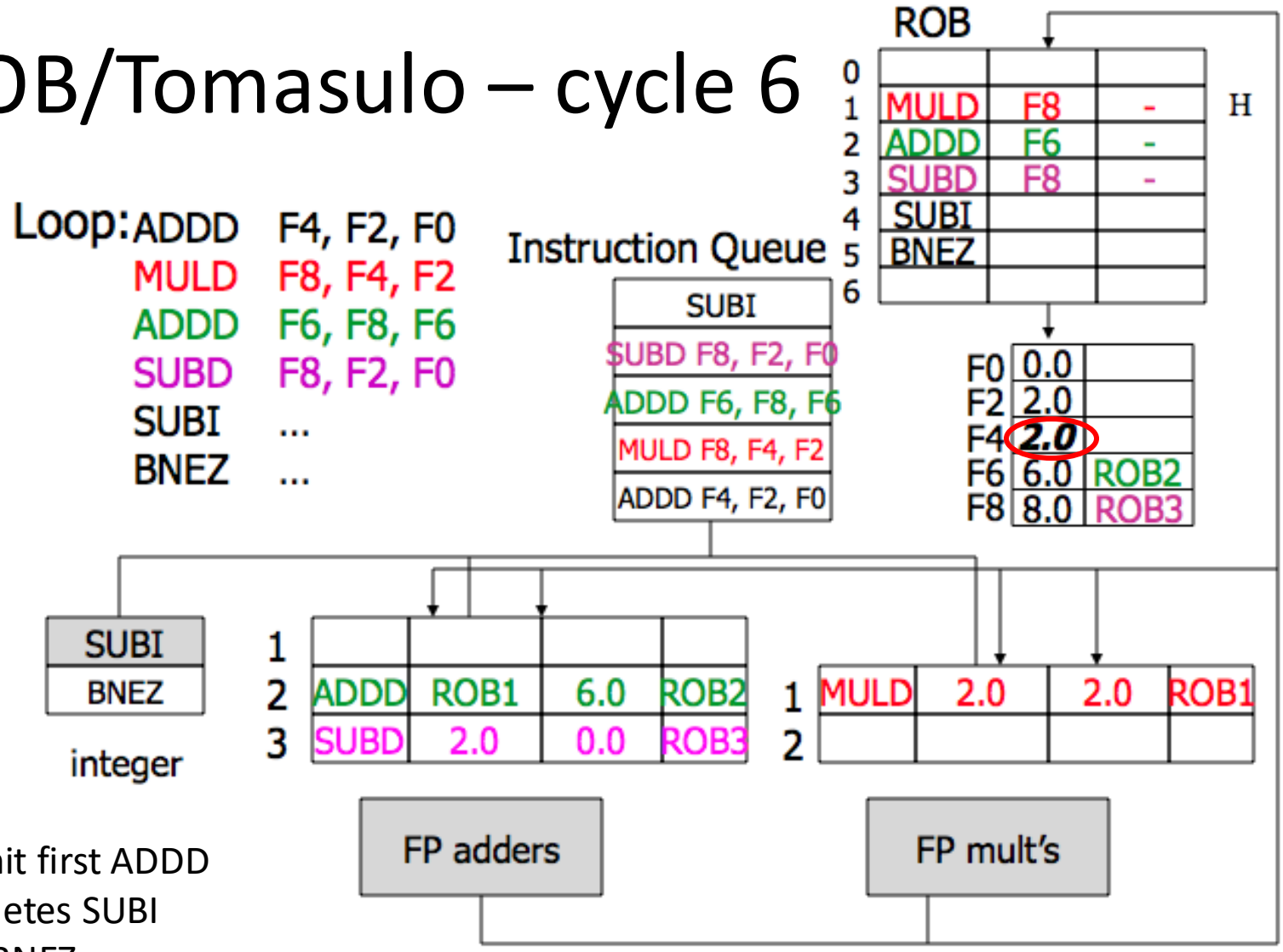
ADDD exe: cc3 ->
ADDD completes

ROB/Tomasulo – cycle 5



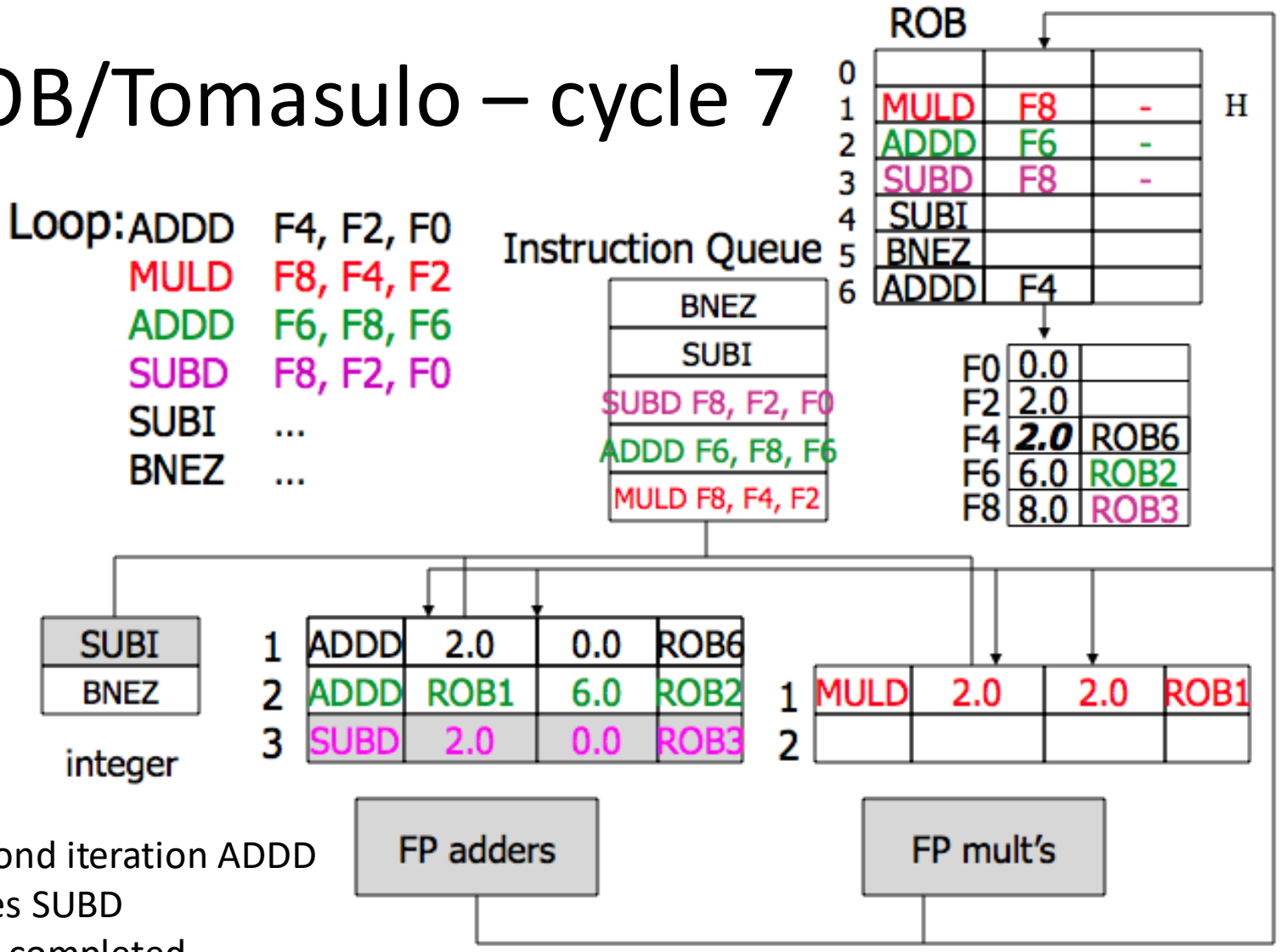
Start SUBD exec
Exec SUBI

ROB/Tomasulo – cycle 6



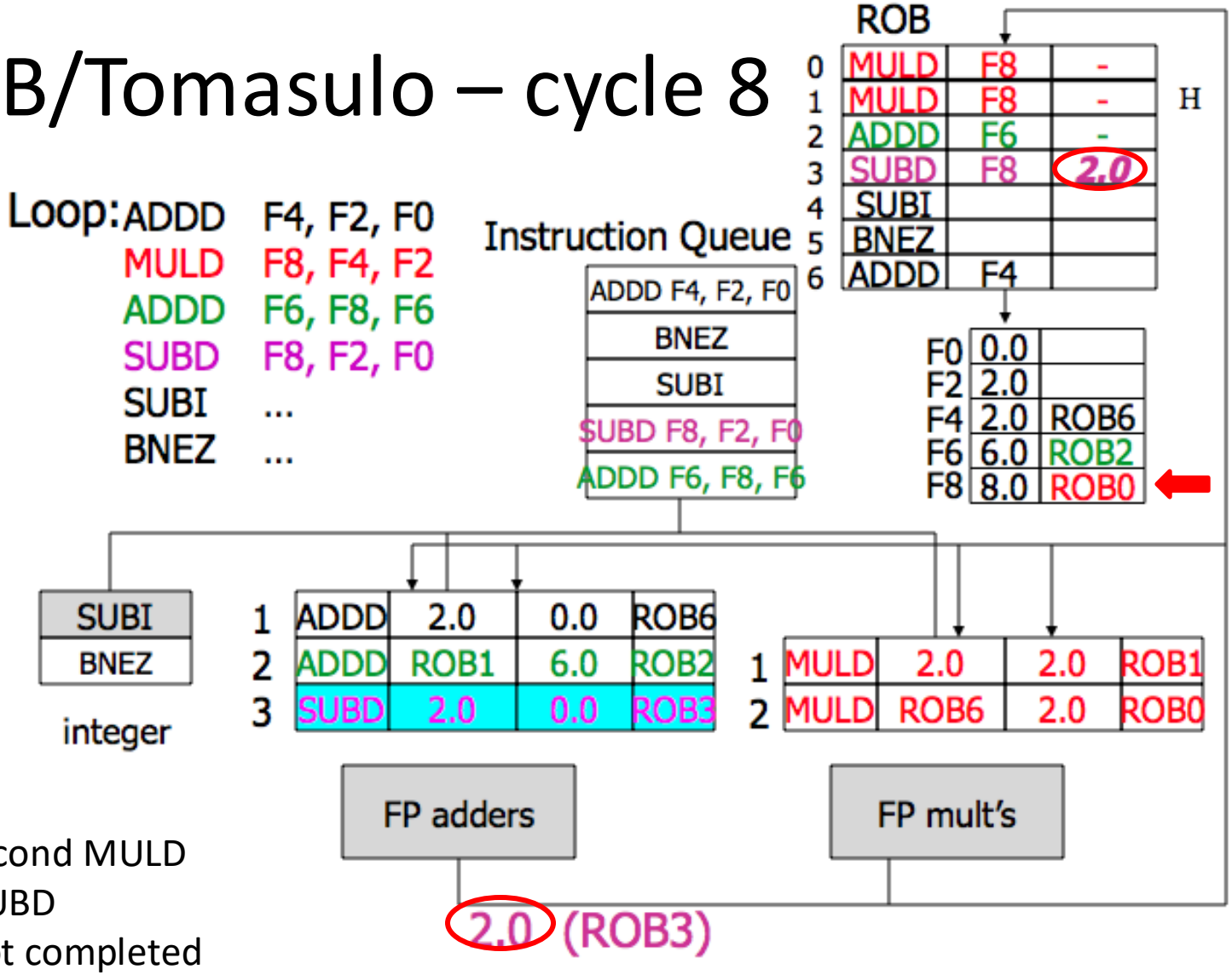
Commit first ADDD
Completes SUBI
Issue BNEZ

ROB/Tomasulo – cycle 7



Issue second iteration ADDD
Completes SUBD
BNEZ not completed

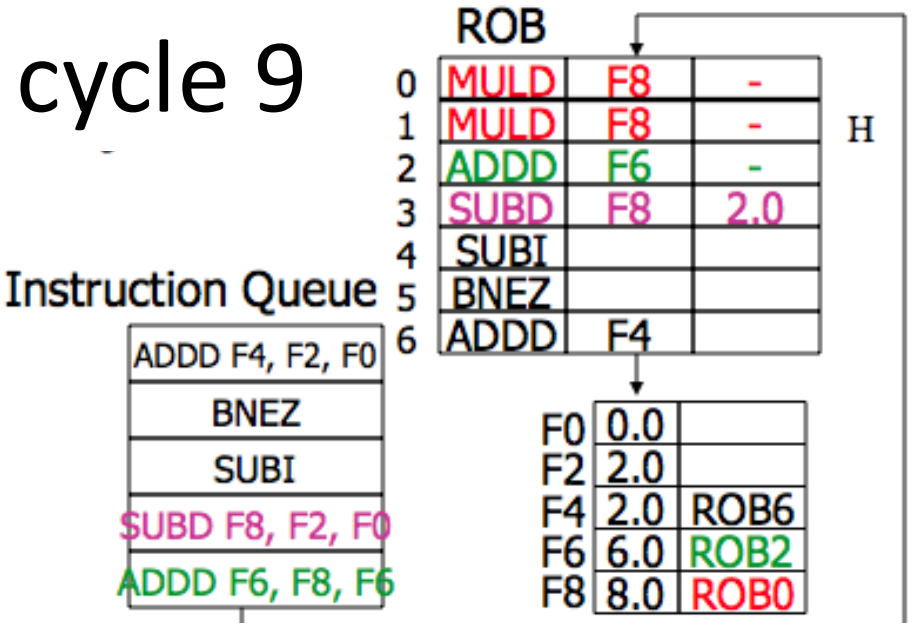
ROB/Tomasulo – cycle 8



Issue second MULD
Write SUBD
BNEZ not completed

ROB/Tomasulo – cycle 9

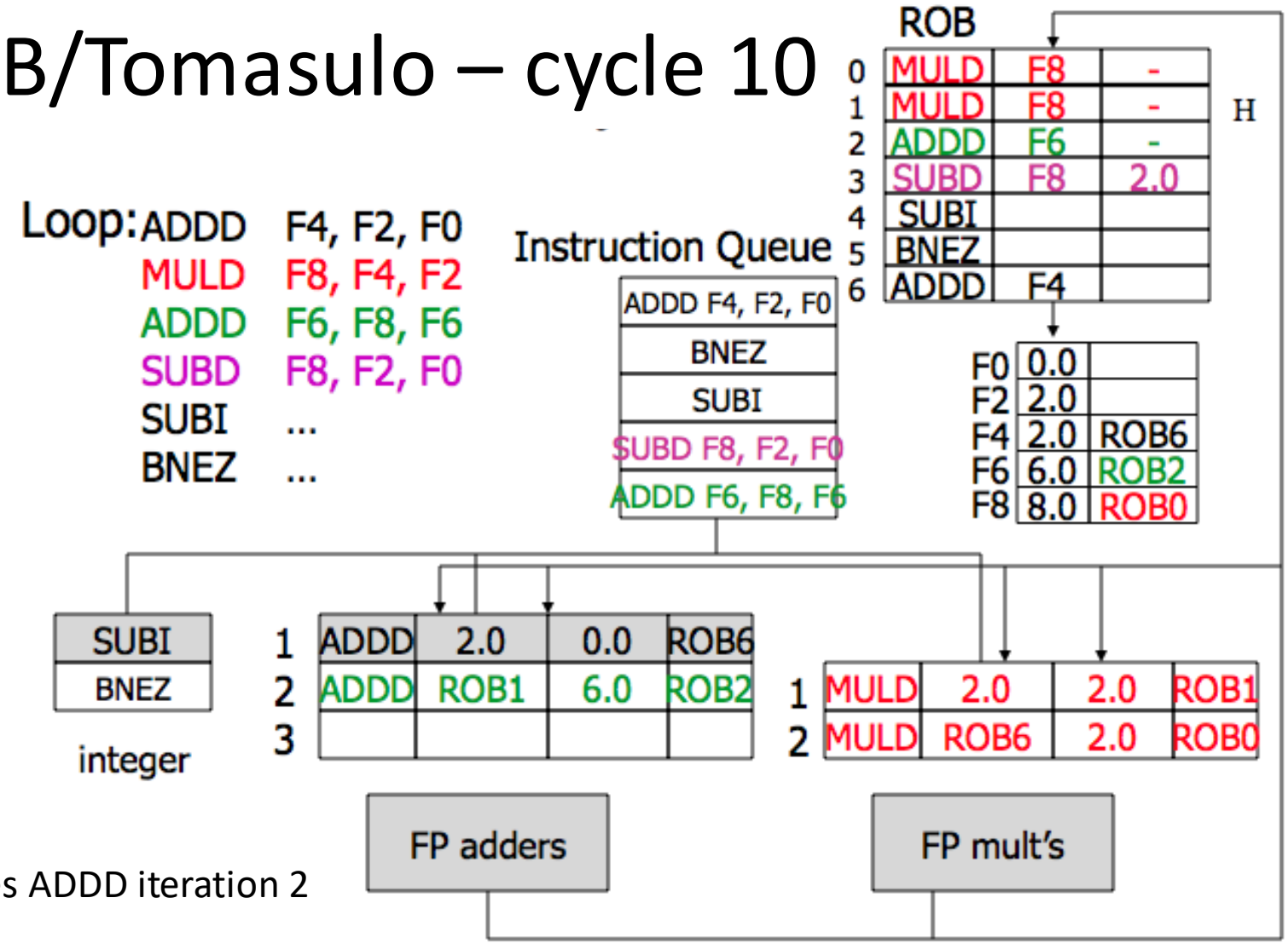
```
Loop: ADDD  F4, F2, F0
      MULD  F8, F4, F2
      ADDD  F6, F8, F6
      SUBD  F8, F2, F0
      SUBI  ...
      BNEZ  ...
```



The 2nd ADDD iteration 2 cannot be issued!

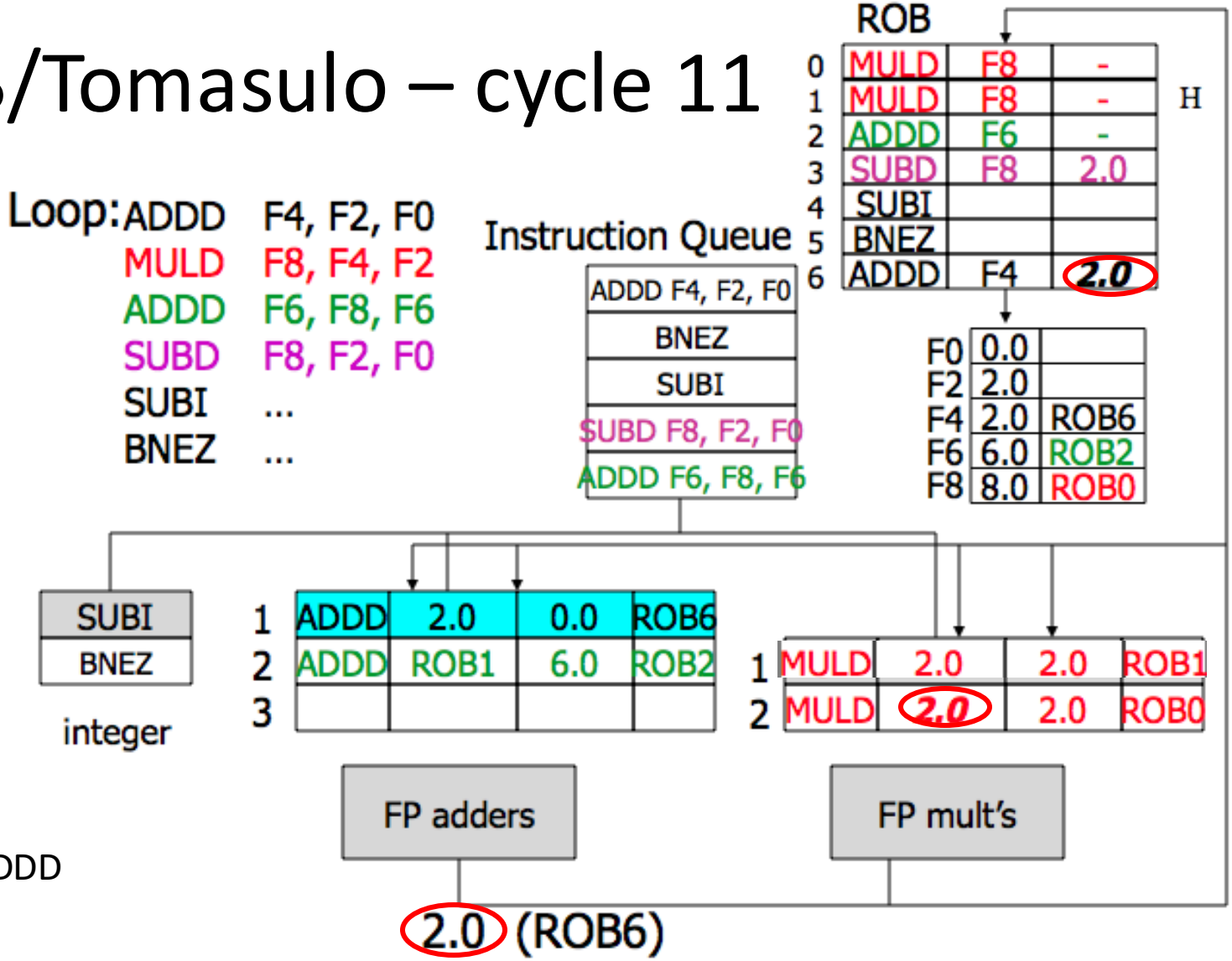
ROB is FULL!

ROB/Tomasulo – cycle 10



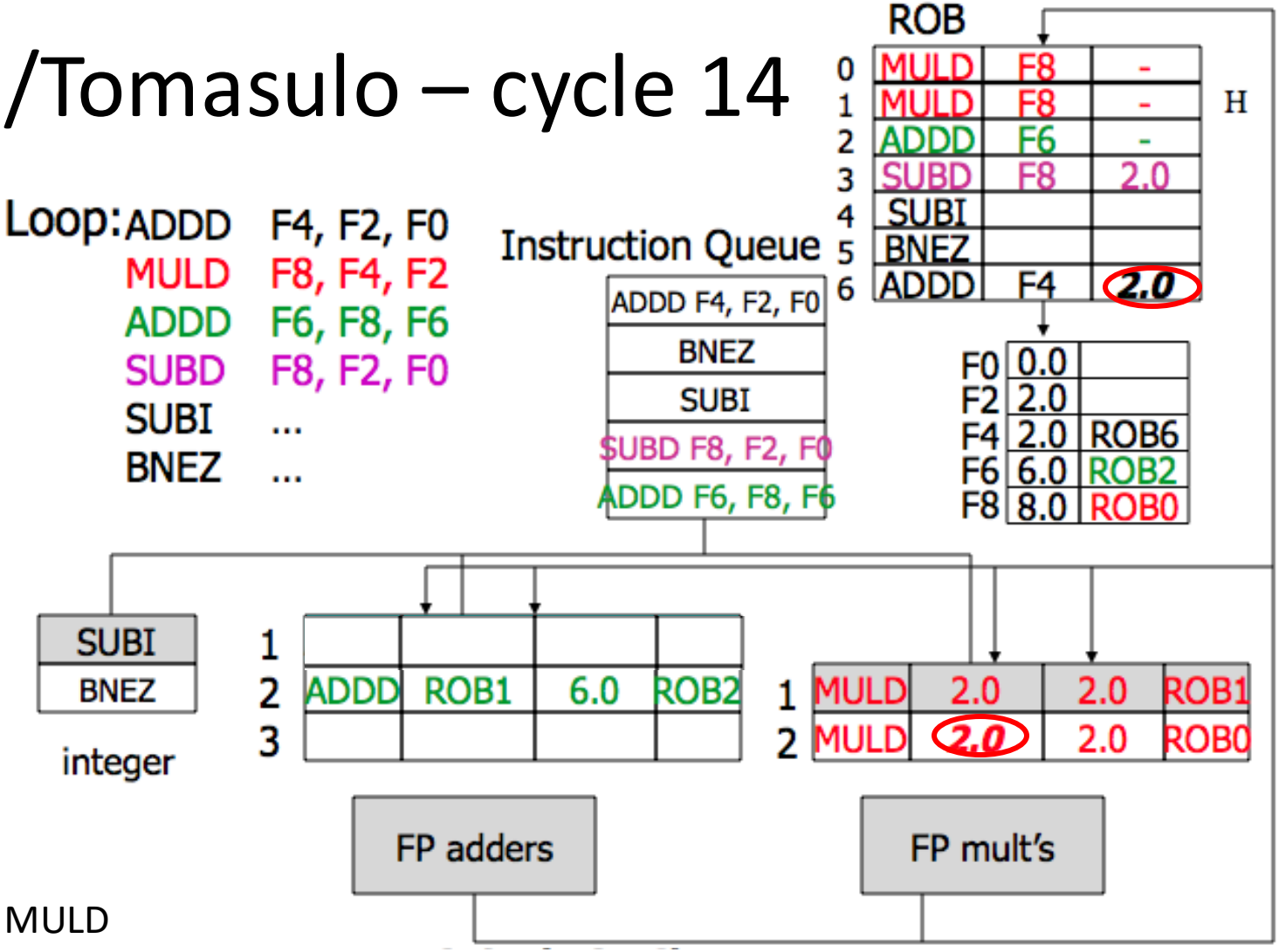
Completes ADDD iteration 2

ROB/Tomasulo – cycle 11



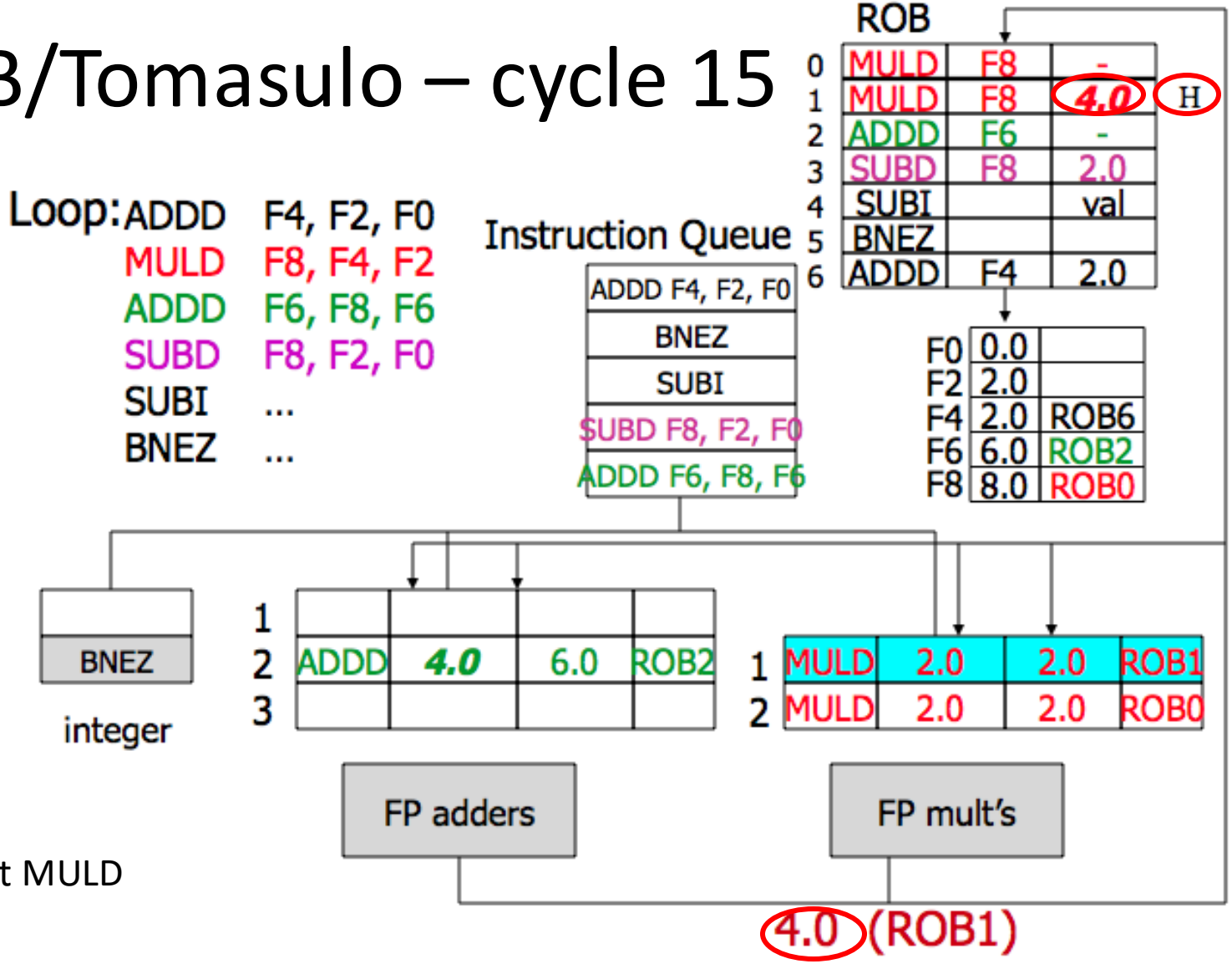
Write ADDD

ROB/Tomasulo – cycle 14



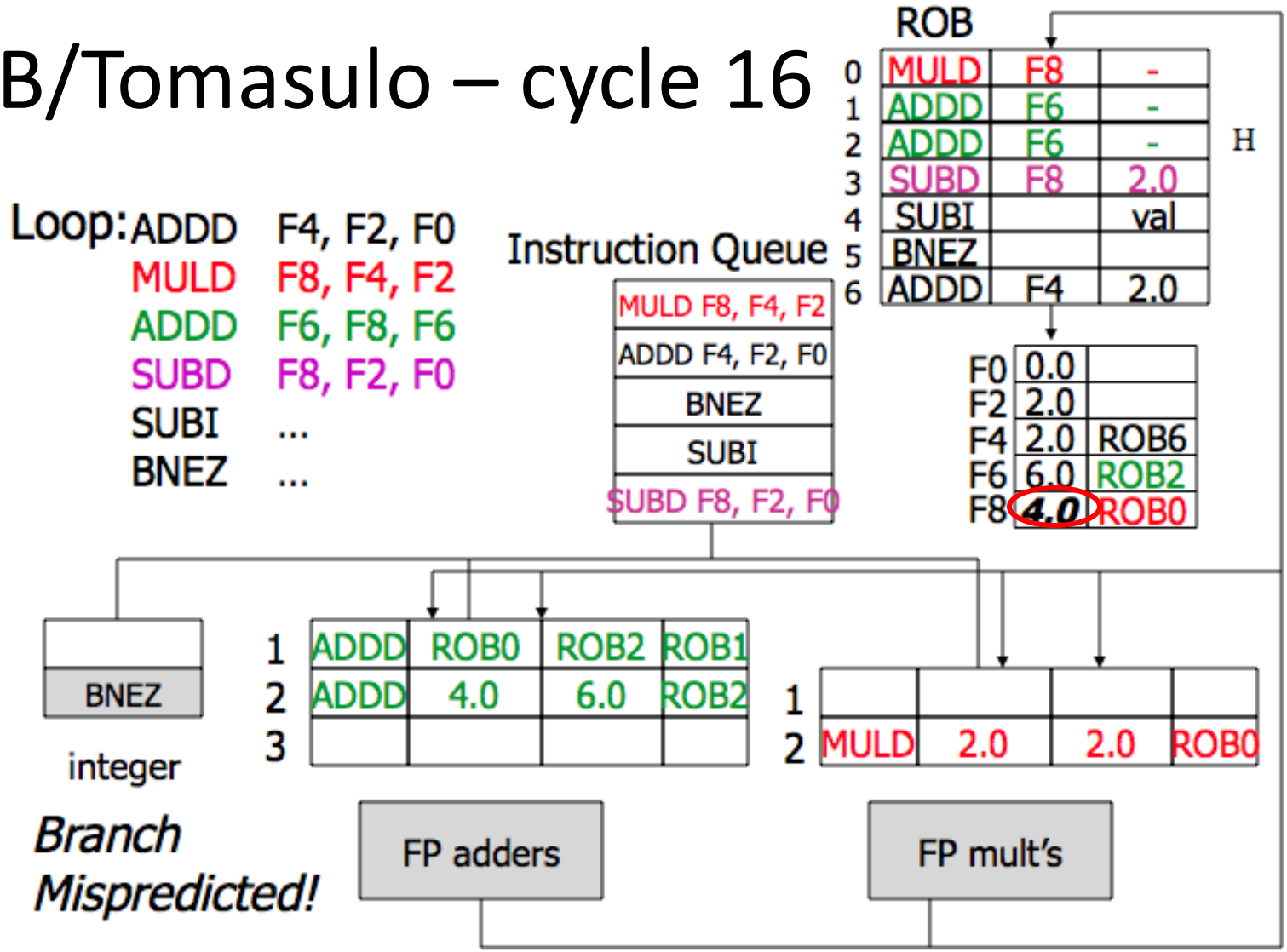
Completes MULD

ROB/Tomasulo – cycle 15



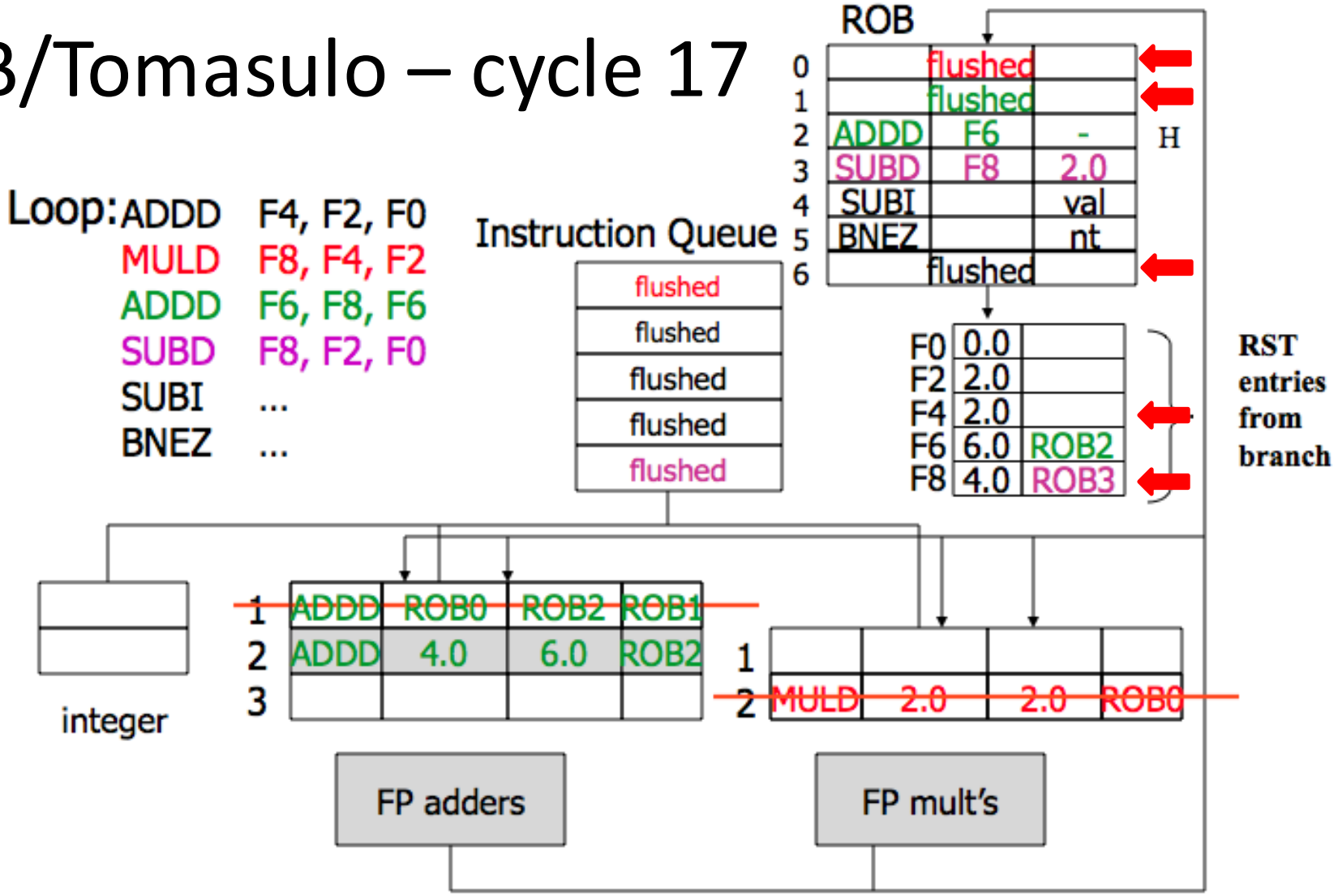
Commit MULD

ROB/Tomasulo – cycle 16

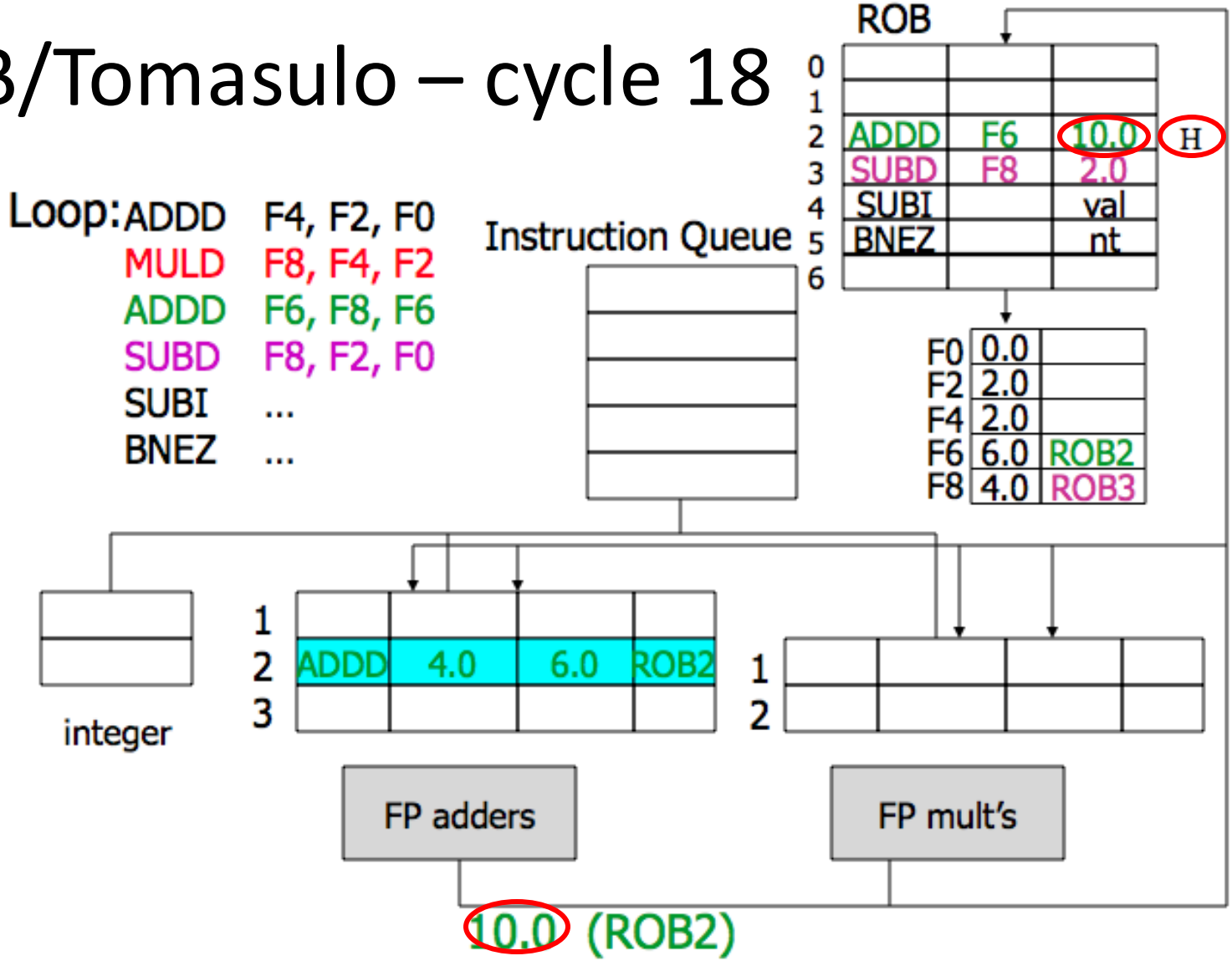


Need to undo 2nd iteration

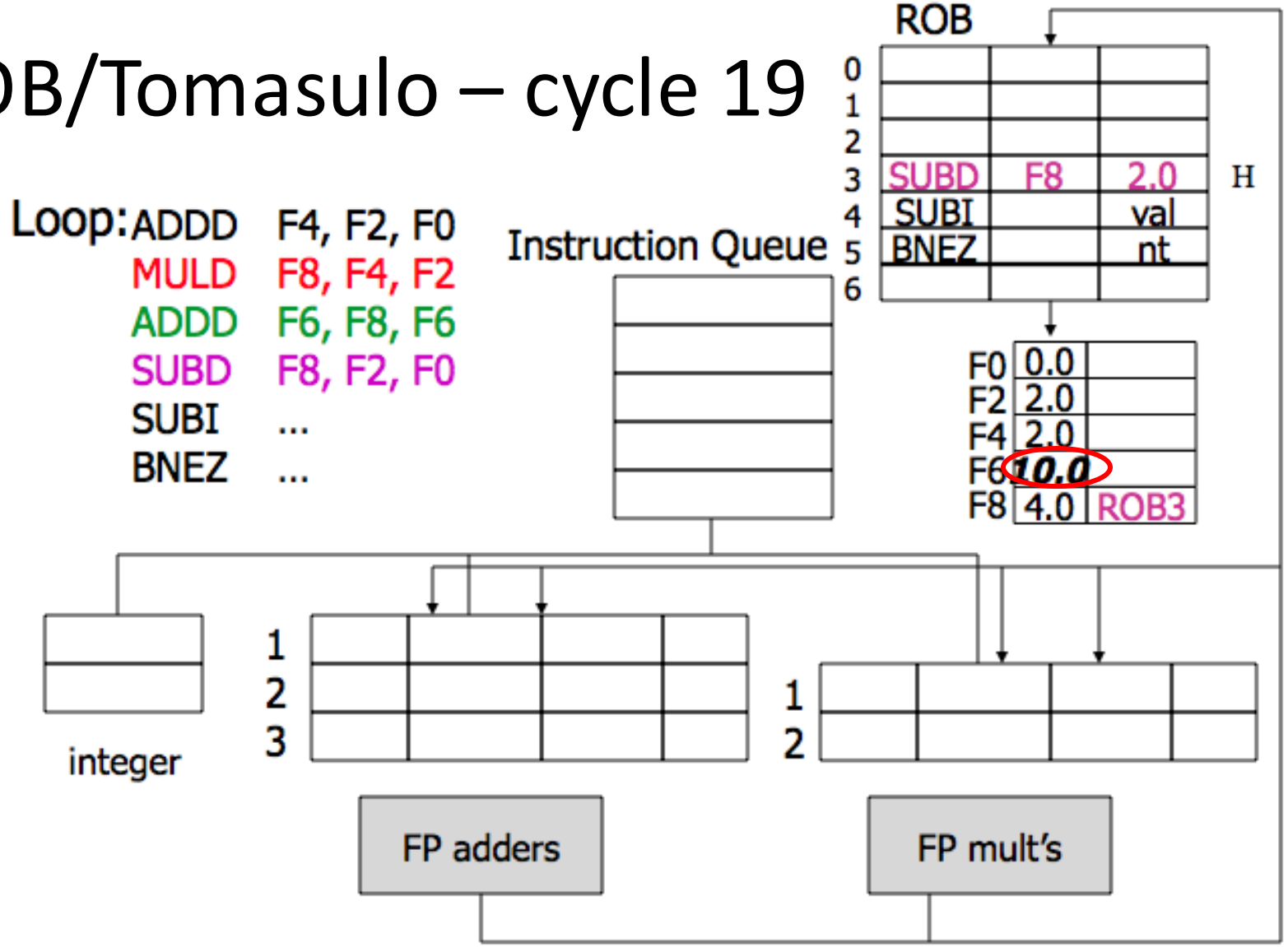
ROB/Tomasulo – cycle 17



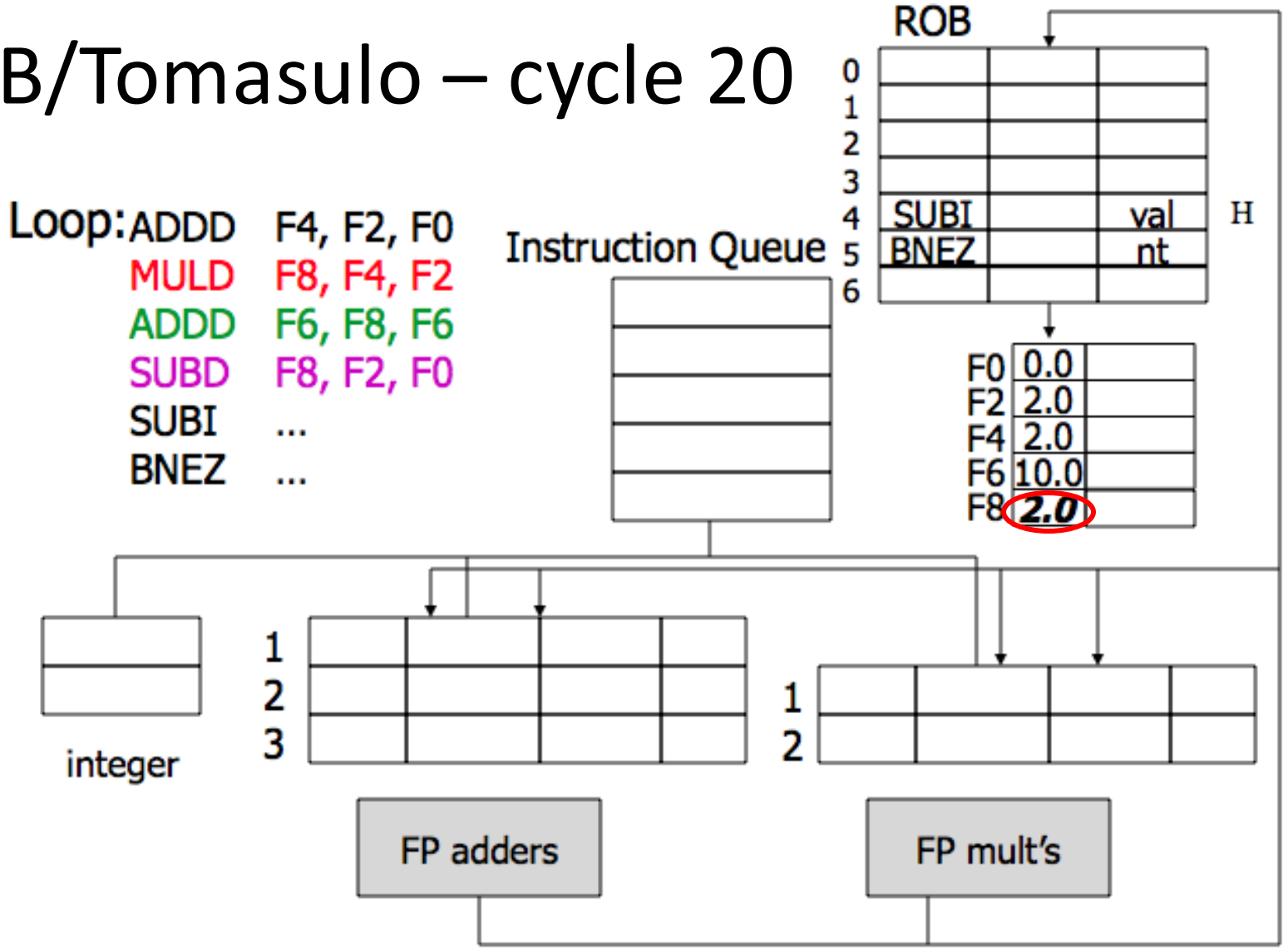
ROB/Tomasulo – cycle 18



ROB/Tomasulo – cycle 19



ROB/Tomasulo – cycle 20



Hardware Based Speculation

Politecnico di Milano
v1