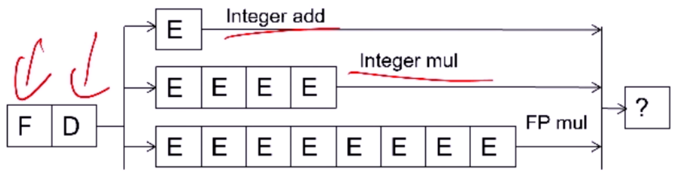
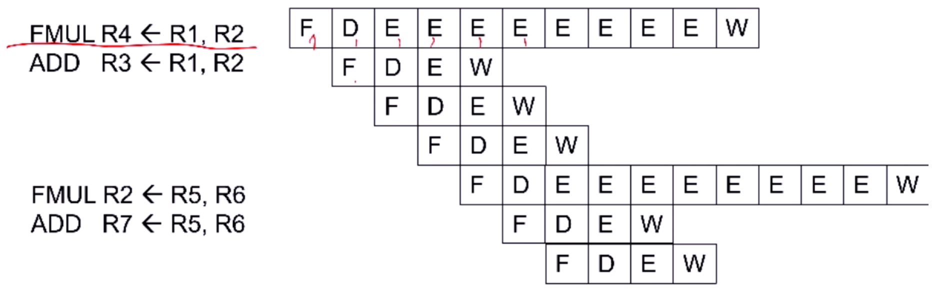
Motivation

- not all instructions need same amount of time when [[Pipelining]]
- idea: mutiple different functional units
  - take different numbers of cycles

    Let independent instructions start execution on a different functional unit before a previous long-latency instruction finishes execution



  -
    - Instructions can take different number of cycles in EXECUTE stage
      - Integer ADD versus FP MULtiply



  -
- Problem
  - instructions might rely on results of previous instructions
  - “contract”
      - Instructions that have been executed up to the PC (program counter) have been executed in the given order
      - Instructions beyond the current value of the PC do not affect the architectural state of the processor
    *

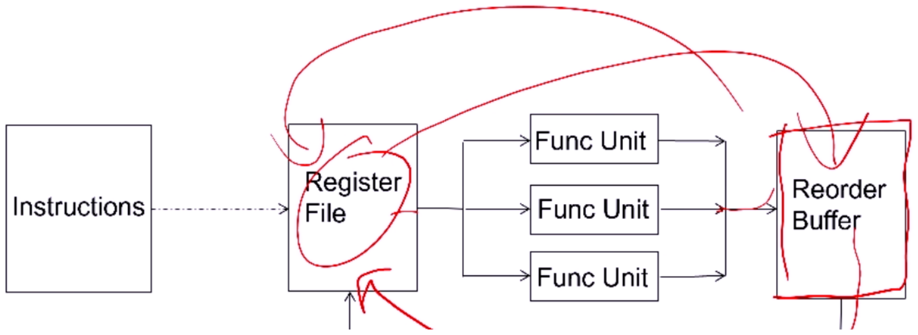
Reorder Buffer (ROB)

- Idea: **Complete** instructions **out-of-order**, but reorder them before making results visible to architectural state

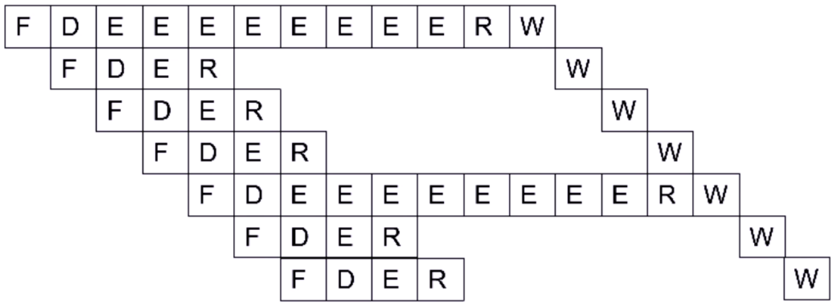
  When instruction is decoded it reserves the next-sequential entry in the ROB

  When instruction completes, it writes result into ROB entry
- When the oldest instruction in the ROB has completed without exceptions, its result is moved to the register file or memory

- Buffers information about all instructions that are decoded but not yet retired/committed
- It needs to store all information that is required to:
  - correctly reorder instructions back into the program order
  - update the architectural state with the instruction's result(s), if instruction can retire without any issues
  - handle an exception/interrupt precisely, if an exception/interrupt needs to be handled before retiring the instruction
- Needs valid bits to keep track of readiness of the result(s) and find out if the instruction has completed execution



- Result first written to ROB on instruction completion
- Result written to register file at commit time



F	D	E	E	E	E	E	E	E	E	R	W						
	F	D	E	R								W					
		F	D	E	R								W				
			F	D	E	R								W			
				F	D	E	E	E	E	E	E	E	E	R	W		
					F	D	E	R								W	
						F	D	E	R								W

Pipeline with ROB + latency overhead +

Efficient ROB Access

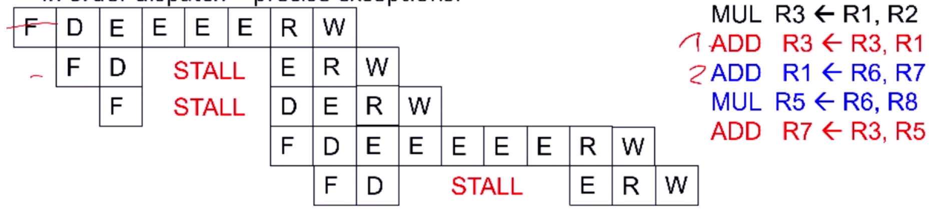
- What if a later instruction needs a value in the reorder buffer?
  - One option: stall the operation → stall the pipeline
  - Better: Read the value from the reorder buffer.

- Access register file first (check if the register is valid)
  - If register not valid, register file stores the ID of the reorder buffer entry that contains (or will contain) the value of the register
  - Mapping of the register to a ROB entry: Register file maps the register to a reorder buffer entry if there is an in-flight instruction writing to the register
- Access reorder buffer next

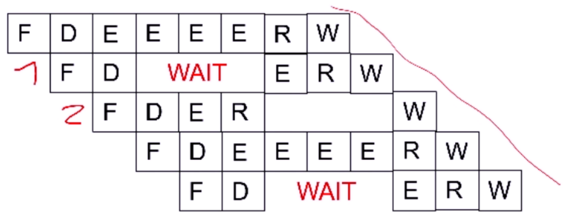
Out-of-Order Dispatch Scheduler

- in-order dispatch has dependencies on prior instructions
- solution: fire instruction, when inputs are ready
- Idea: Move the dependent instructions out of the way of independent ones (such that independent ones can execute)
  - Rest areas for dependent instructions
- Monitor the source "values" of each instruction in the resting area
- When all source "values" of an instruction are available, "fire" (i.e. dispatch) the instruction
  - Instructions dispatched in dataflow (not control-flow) order

- In order dispatch + precise exceptions:



- Out-of-order dispatch + precise exceptions:


F D E E E E R W
1 F D WAIT E R W
2 F D E R W
F D E E E E R W
F D WAIT E R W