Fiscal Year 2021

Ver. 2022-01-17a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

8. Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation

www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 14:20-16:00, Thr 14:20-16:00

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Hardware register renaming (last lecture)

- Logical registers (architectural registers) which are ones defined by ISA
  - \$0, \$1, … \$31
- Physical registers
  - Assuming plenty of registers are available, p0, p1, p2, …
- A processor renames (converts) each logical register to a unique physical register dynamically

Typical instruction pipeline of scalar processor

IF	ID	EX	MEM	WB

Typical instruction pipeline of high-performance superscalar processor

IF	ID	Renaming	Dispatch	Issue	Execute	Commit	Retire

Out-of-order execution

- In in-order execution model, all instructions are executed in the order that they appear.
  This can lead to unnecessary stalls.
  - Instruction (3) stalls waiting for insn (2) to go first, even though it does not have a data dependence.
- With out-of-order execution,
  - Using register renaming to eliminate output dependence and antidependence, just having true data dependence
  - insn (3) is allowed to be executed before the insn (2)
    - Scoreboarding (CDC6600 in 1964)
    - Tomasulo algorithm
      (IBM System/360 Model 91 in 1967)

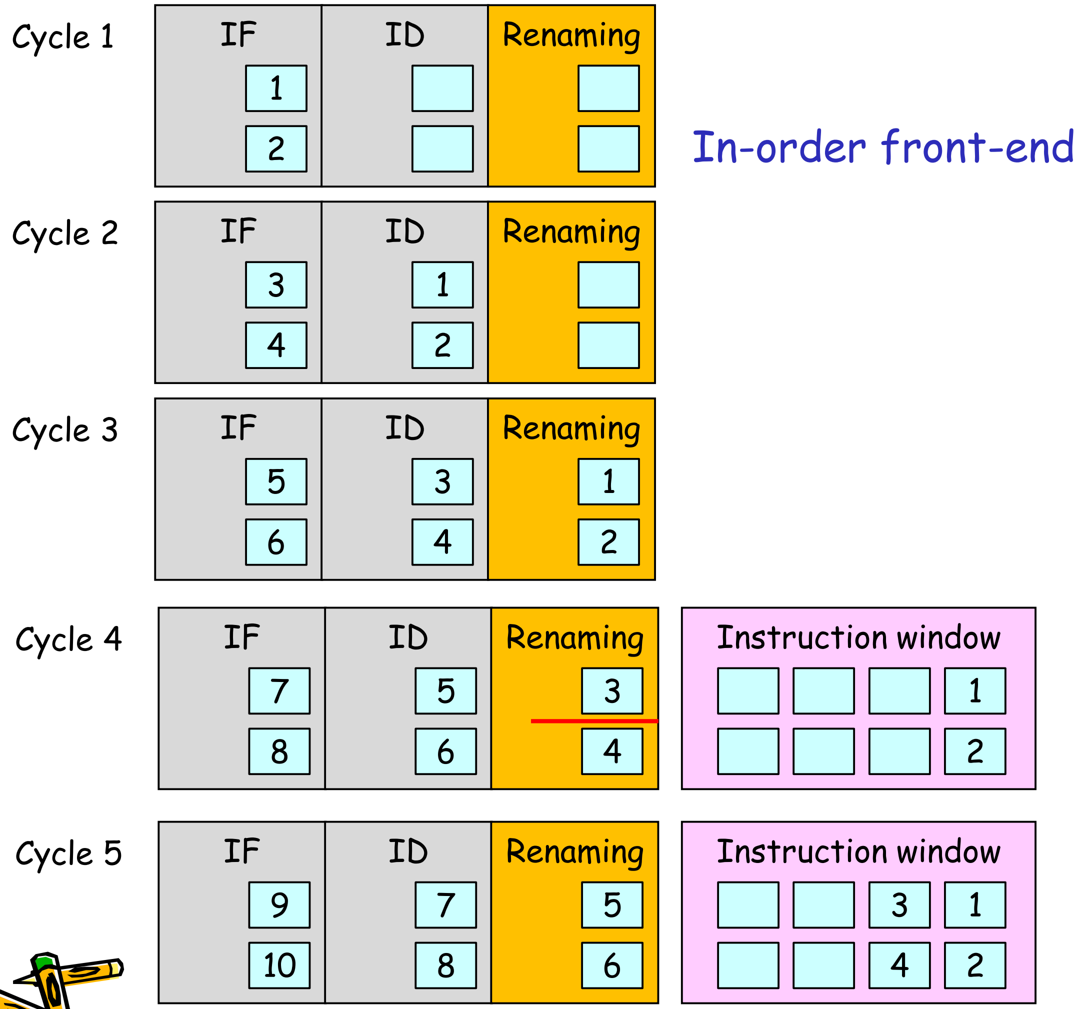
$$\begin{tikzcd}
 & (1) \arrow[d] \\
(3) \arrow[dr] & (2) \arrow[d] \\
 & (4)
\end{tikzcd}$$

```
R3  =  R3  x  R5   (1)
R4  =  R3  +  1    (2)
R3  =  R5  +  2    (3)
R7  =  R3  +  R4   (4)
```

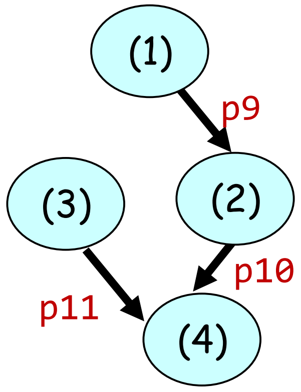
Data flow graph

The key idea for OoO execution (1/3)

- In-order front-end, OoO execution core, in-order retirement using instruction window and reorder buffer (ROB)



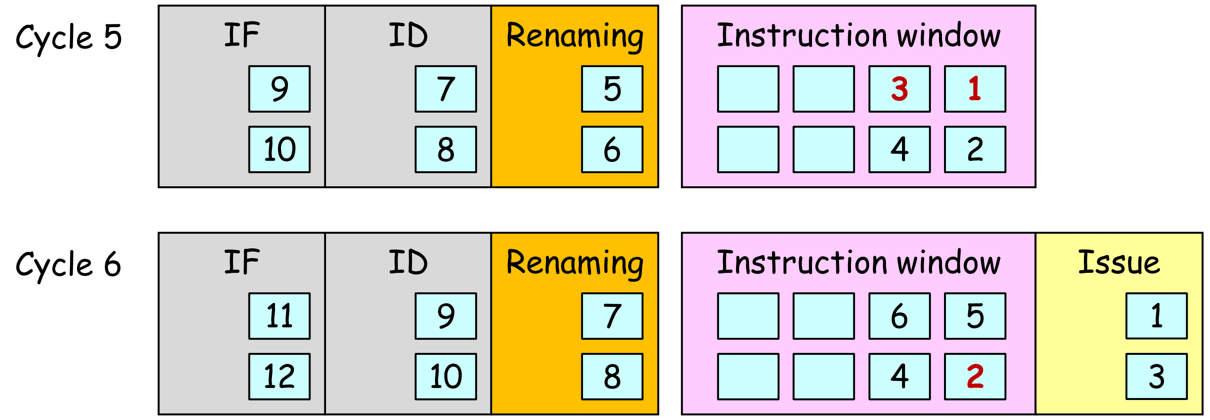
```
I1:  sub p9,p1,p2
I2:  add p10,p9,p3
I3:  or  p11,p4,p5
I4:  and p12,p10,p11
```



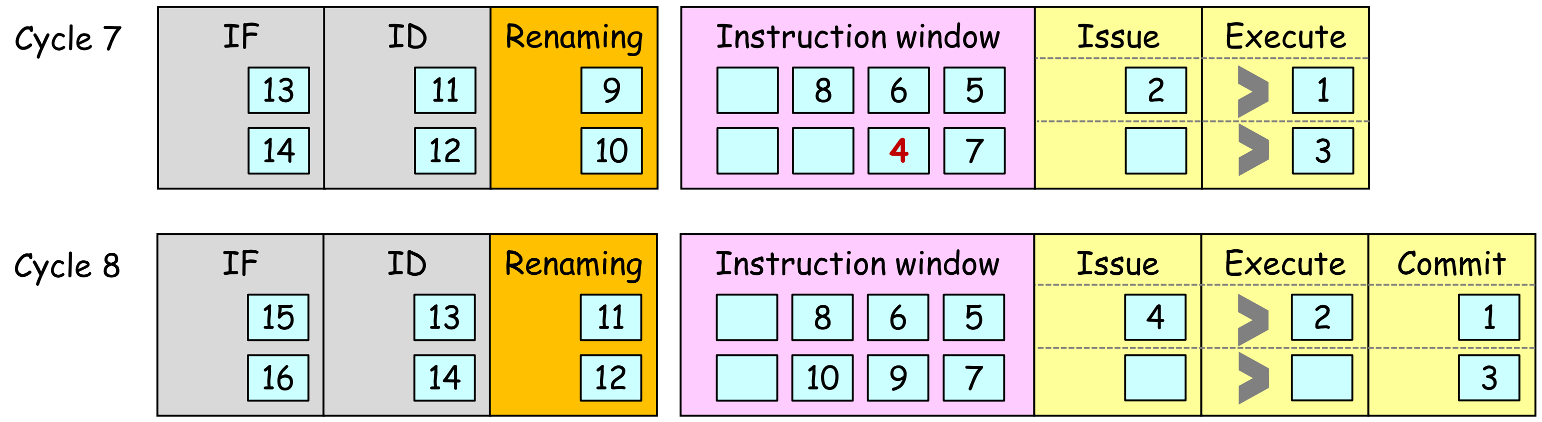
Data flow graph

The key idea for OoO execution (2/3)

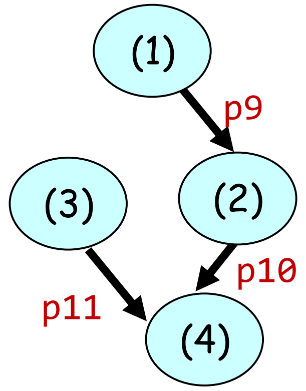
- In-order front-end, OoO execution core, in-order retirement using instruction window and reorder buffer (ROB)



We assume that I1 and I3 can be issued at cycle 6 by dependence.



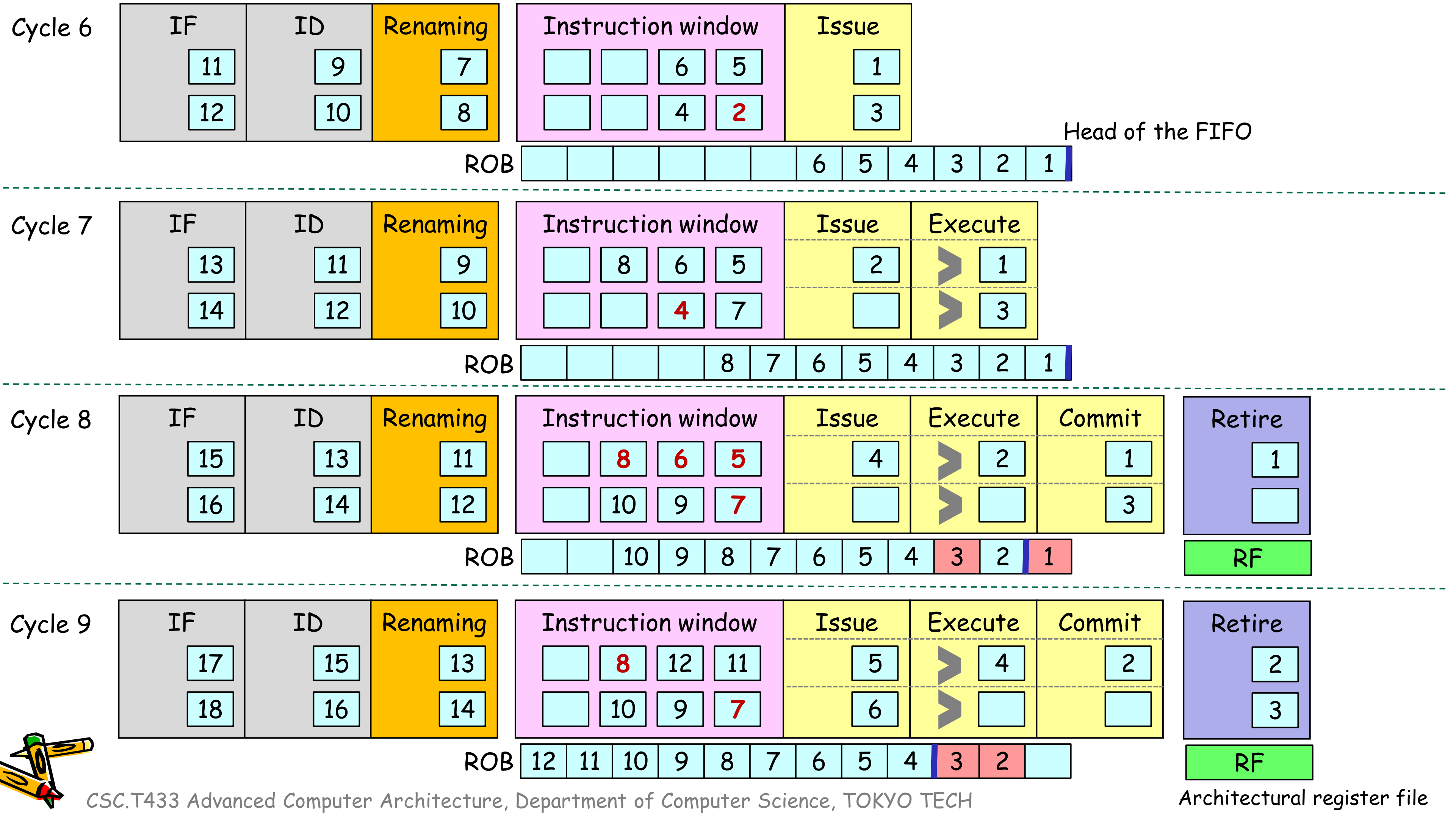
```
I1: sub p9,p1,p2
I2: add p10,p9,p3
I3: or  p11,p4,p5
I4: and p12,p10,p11
```



Data flow graph

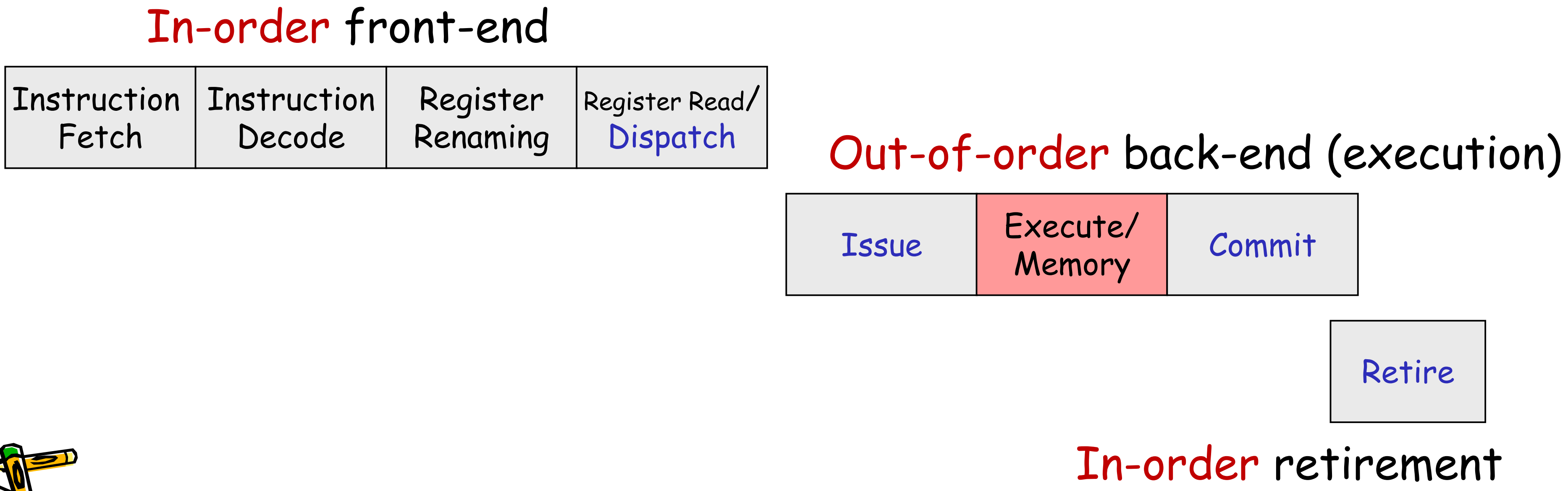
The key idea for OoO execution (3/3)

- In-order front-end, OoO execution core, in-order retirement using instruction window and reorder buffer (ROB)



Instruction pipeline of OoO execution processor

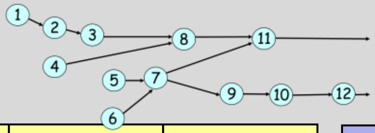
- Allocating instructions to instruction window is called dispatch
- Issue or fire wakes up instructions and their executions begin
- In commit stage, *the computed values are written back to ROB*
- The last stage is called retire or graduate. The completed consecutive instructions can be retired.
  The result is written back to register file (*architectural register file*) using a logical register number.



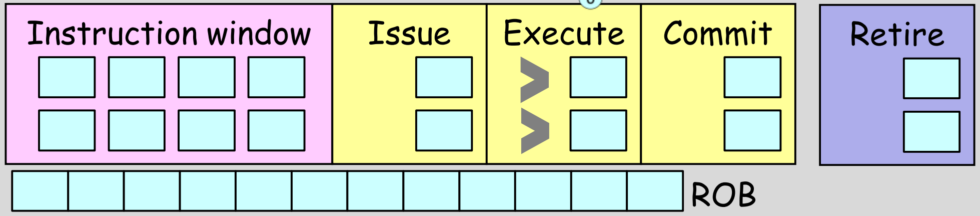
Exercise: OoO execution

- Draw the cycle by cycle processing behavior of these 12 instructions
  - wakeup
  - select

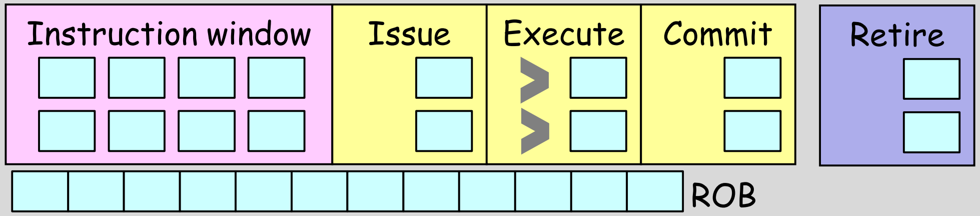
$$\begin{tikzcd}
1 \arrow[dr] & & & & & & & & \\
 & 2 \arrow[dr] & & & & & & & \\
 & & 3 \arrow[rr] & & 8 \arrow[rr] & & 11 \arrow[rr] & & {} \\
 & 4 \arrow[urrr] & & & & & & & \\
 & & 5 \arrow[r] & 7 \arrow[uurrr] \arrow[drr] & & & & & \\
 & & & & & 9 \arrow[r] & 10 \arrow[r] & 12 \arrow[r] & {} \\
 & & 6 \arrow[uur] & & & & & &
\end{tikzcd}$$



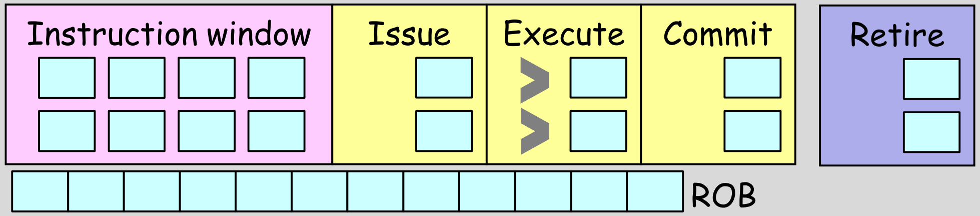
Cycle 1



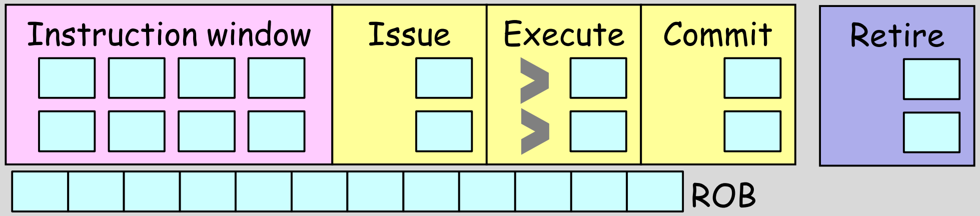
Cycle 2



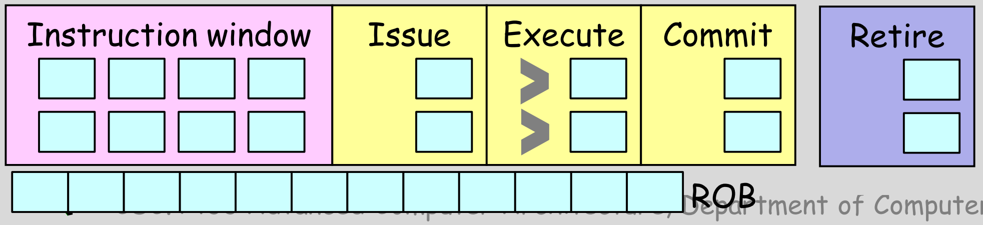
Cycle 3

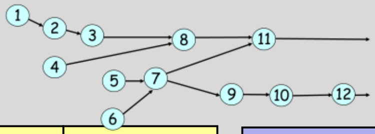


Cycle 4

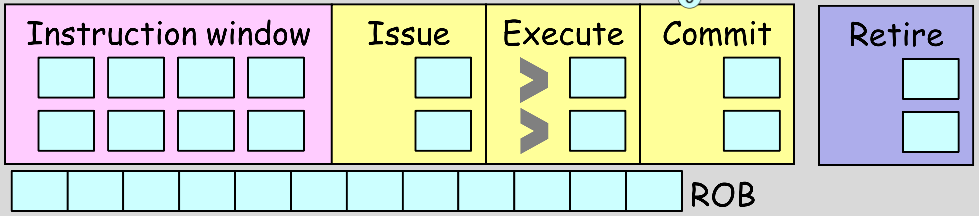


Cycle 5

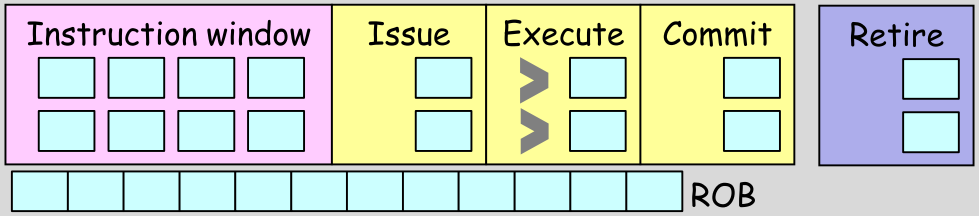




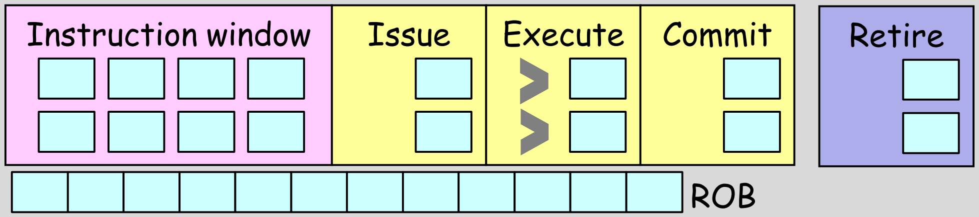
Cycle 6



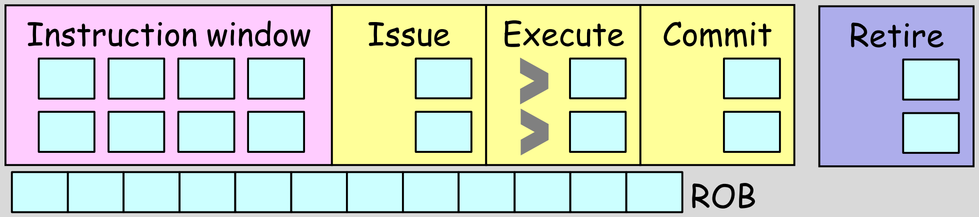
Cycle 7



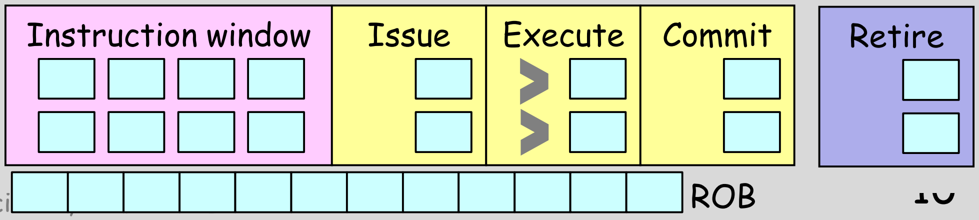
Cycle 8



Cycle 9

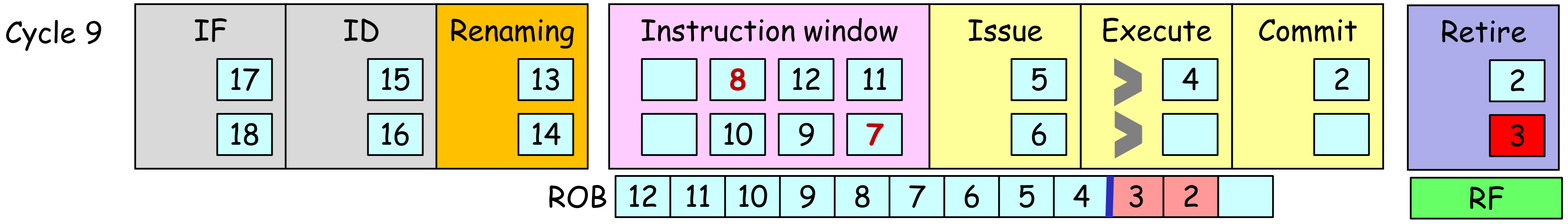


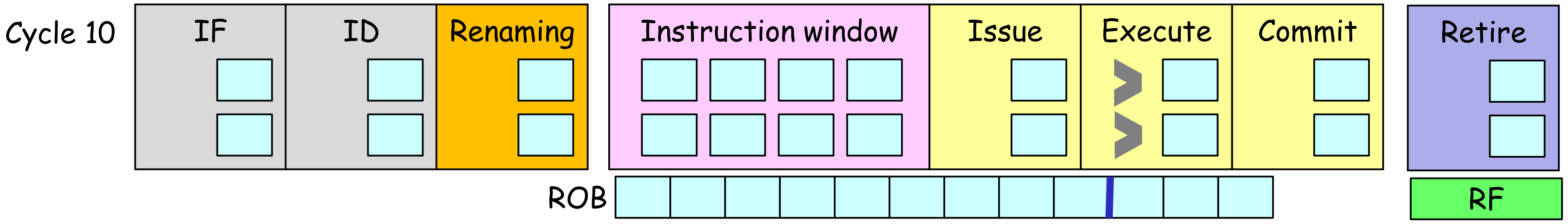
Cycle 10



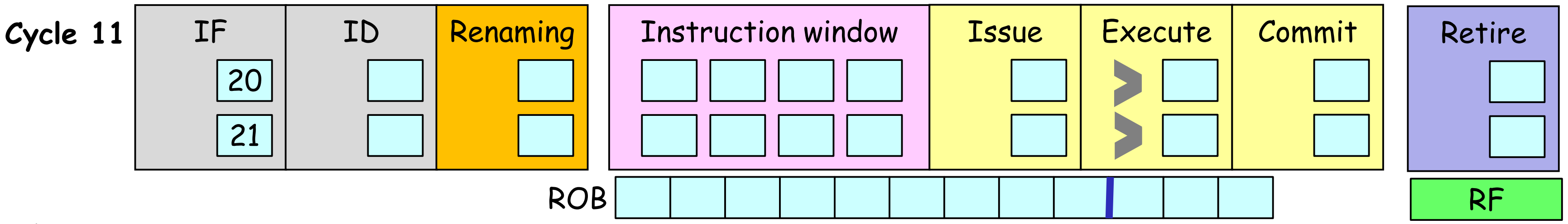
Prediction miss and recovery

- Assume that instruction 3 is a miss predicted branch and its target insn is 20
- When insn 3 is retired, it recovers by flushing all instructions and restart
- Register file (and PC) has the architecture state after insn 3 is executed





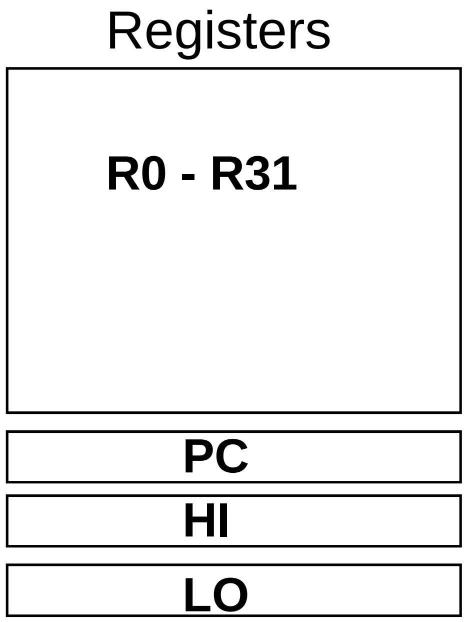
Recovery by flushing instructions on the wrong path (may take several cycles)



Restart by fetching instructions using the correct PC

MIPS R3000 Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

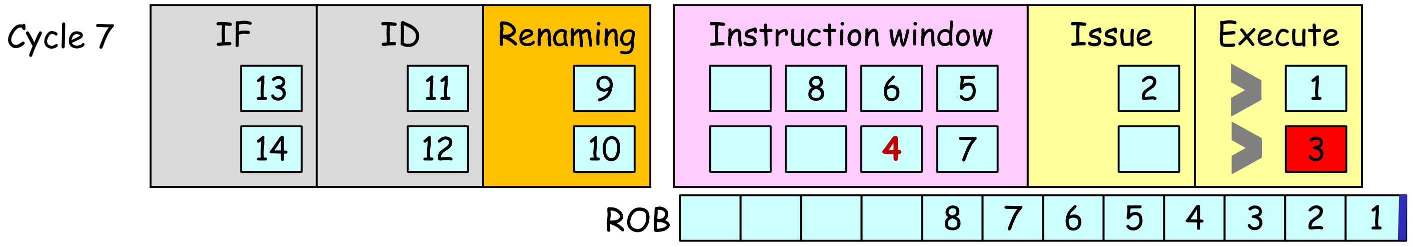


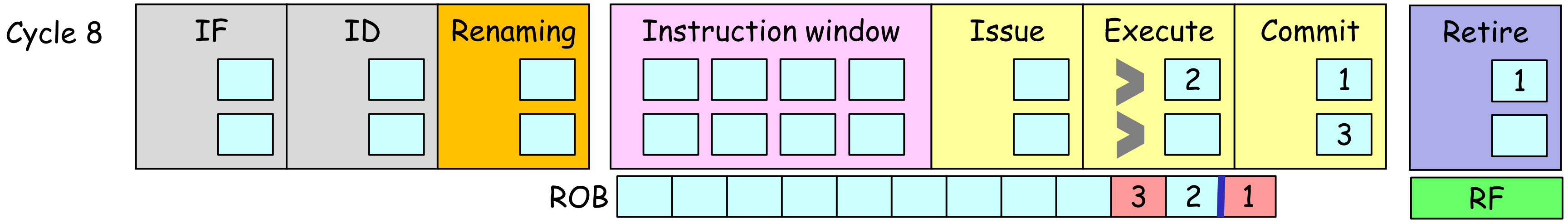
3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	shamt	funct	R format
OP	rs	rt	immediate			I format
OP	jump target (immediate)					J format

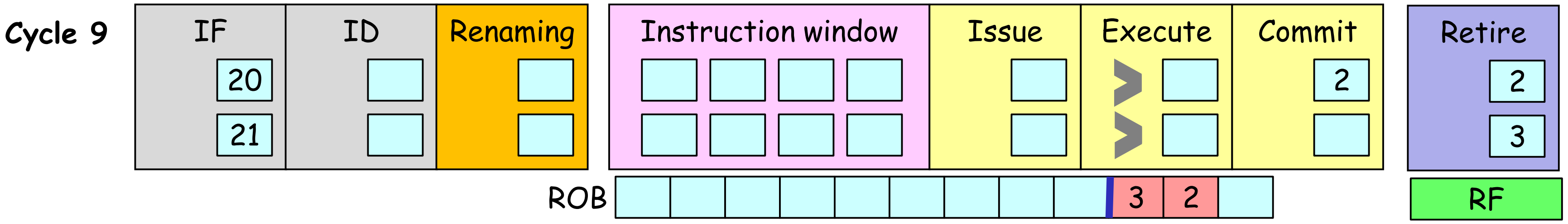
Branch prediction miss and aggressive recovery

- Instruction 3 is a miss predicted branch and its target insn is 20
- When insn 3 is executed, it recovers by flushing instructions after insn 3 and restarts





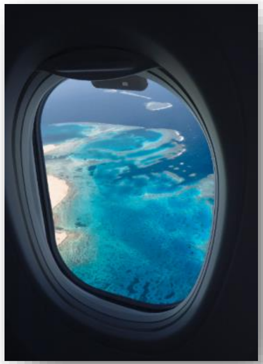
Recovery by flushing instructions on the wrong path (may takes several cycles)

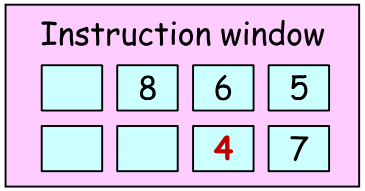


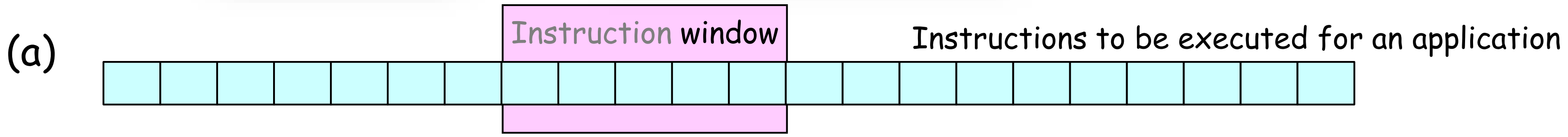
Restart by fetching instructions using the correct PC

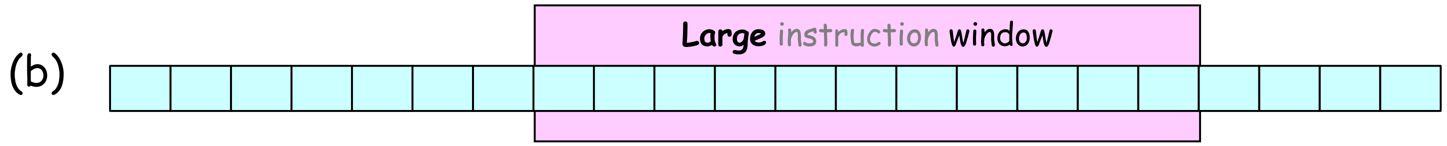
Aside: What is a window?

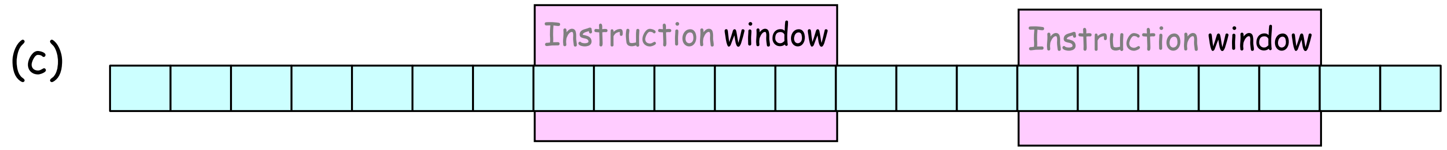
- A window is a space in the wall of a building or in the side of a vehicle, which has glass in it so that light can come in and you can see out. (Collins)





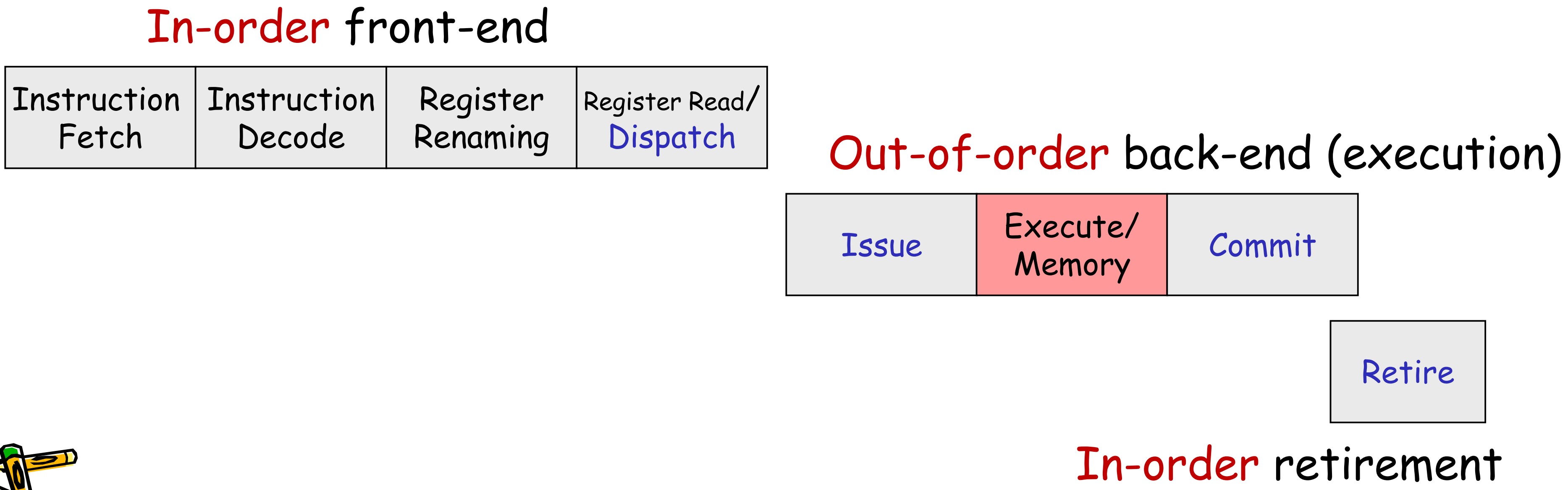




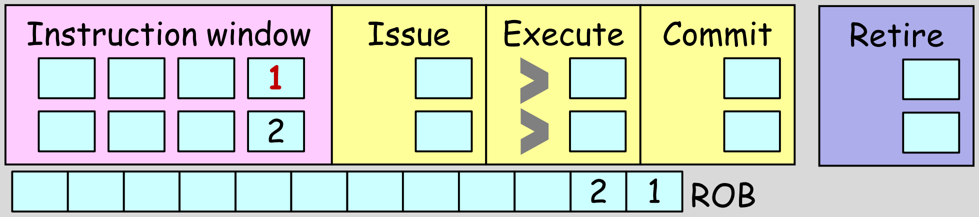


Instruction pipeline of OoO execution processor

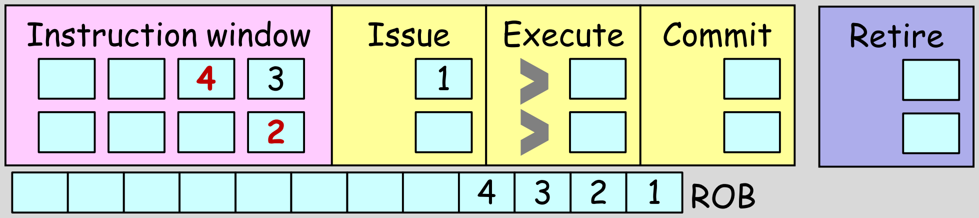
- Allocating instructions to instruction window is called dispatch
- Issue or fire wakes up instructions and their executions begin
- In commit stage, *the computed values are written back to ROB*
- The last stage is called retire or graduate. The completed consecutive instructions can be retired.
  The result is written back to register file (*architectural register file*) using a logical register number.



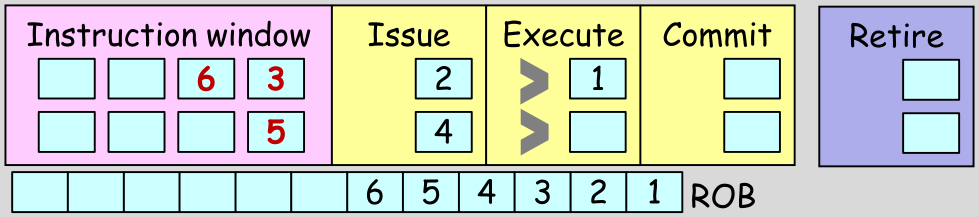
Cycle 1



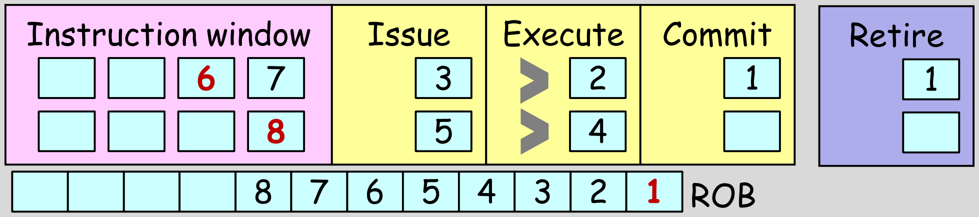
Cycle 2



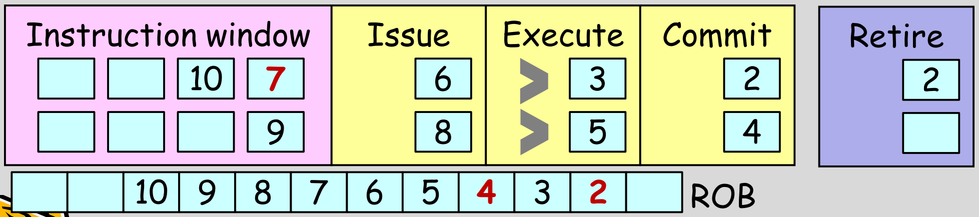
Cycle 3



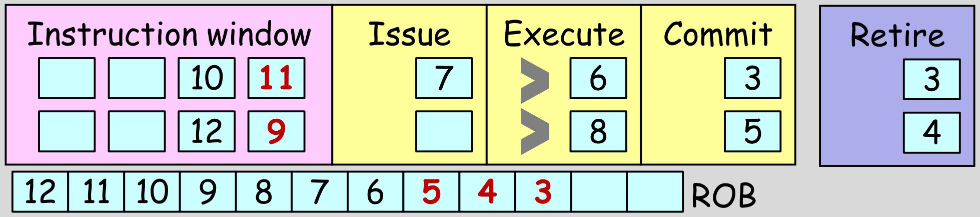
Cycle 4



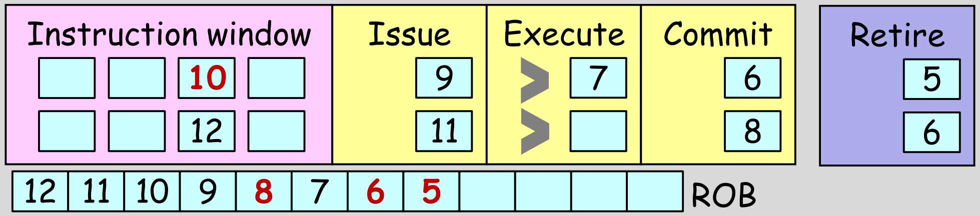
Cycle 5



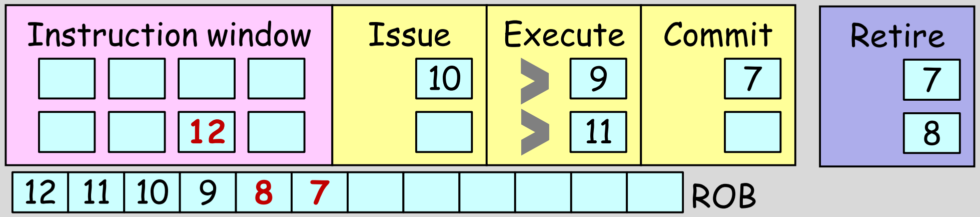
Cycle 6



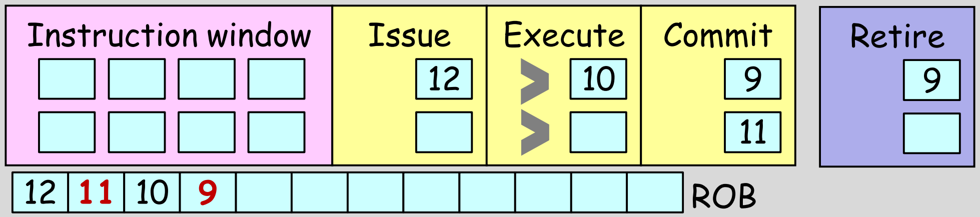
Cycle 7



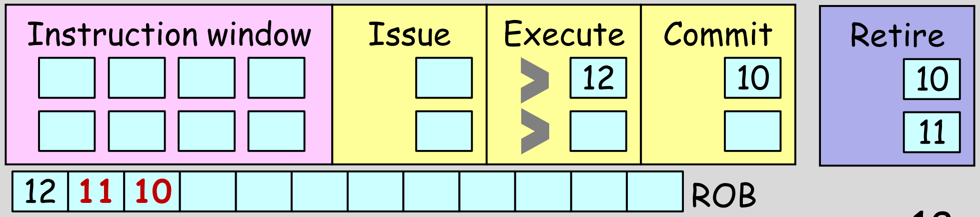
Cycle 8



Cycle 9



Cycle 10


Instruction window
Issue
Execute
12
Commit
10
Retire
10
11
12 11 10
ROB