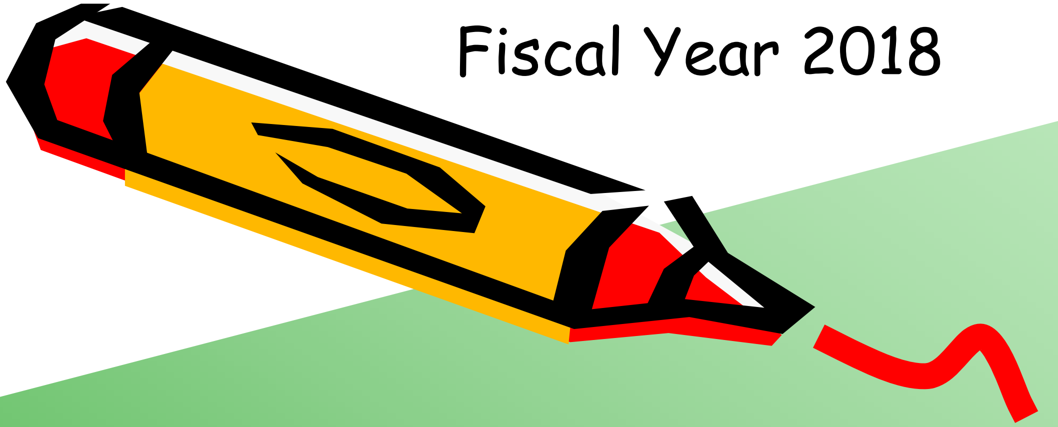
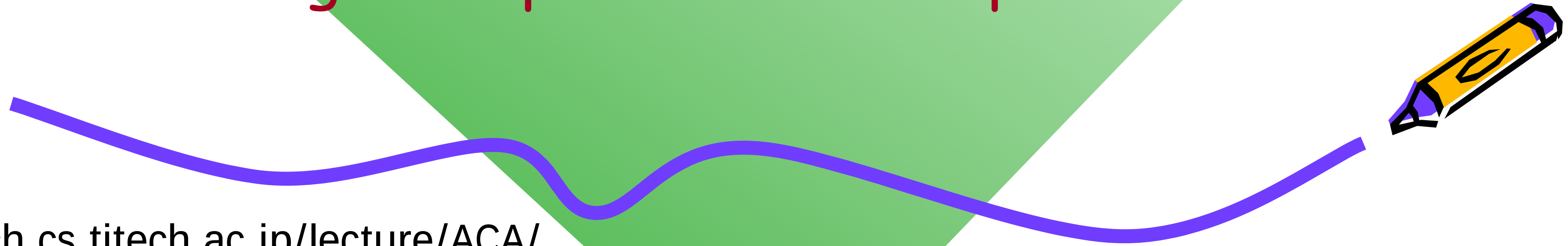
Fiscal Year 2018



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

Advanced Computer Architecture

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



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

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

Hardware register renaming

- 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

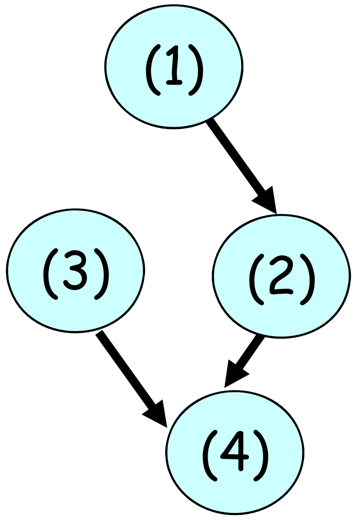
IF	ID	EX	MEM	WB

Typical instruction pipeline of high-performance superscalar processor

IF	ID	Renaming	Dispatch	Issue	Execute	Commit	Retire

Out-of-order execution

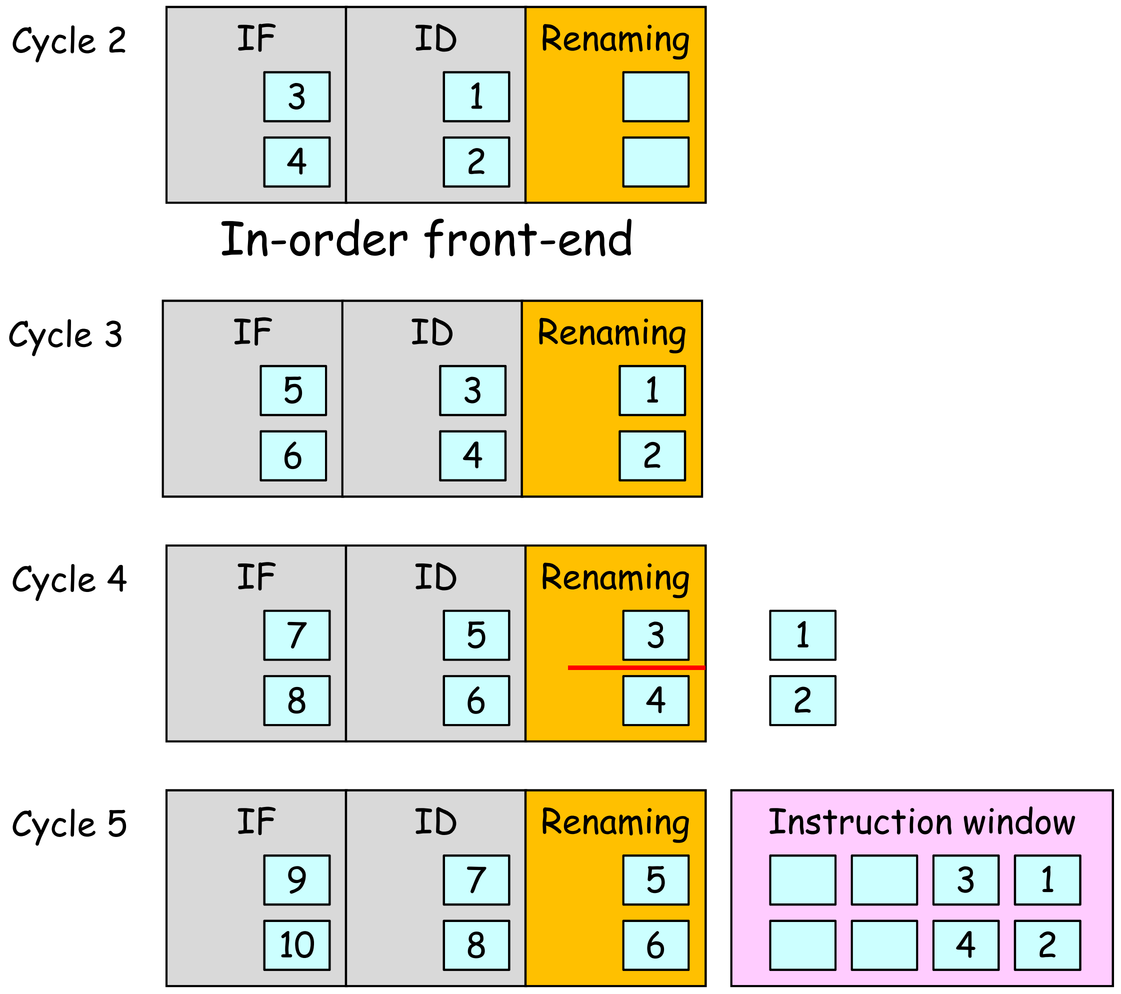
- In in-order execution model, all instructions executed in the order that they appear. This can lead to unnecessary stalls.
  - Instruction (3) stalls waiting for insn (2) to go first, even though it does not have a data dependence.
- Using register renaming to eliminate output dependence and antidependence,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


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

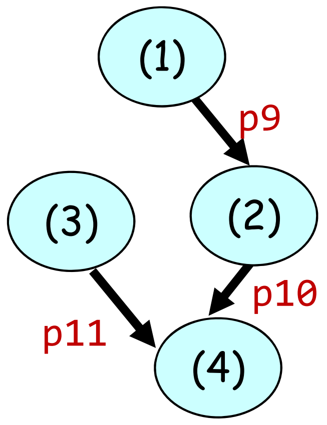
Data flow graph

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)



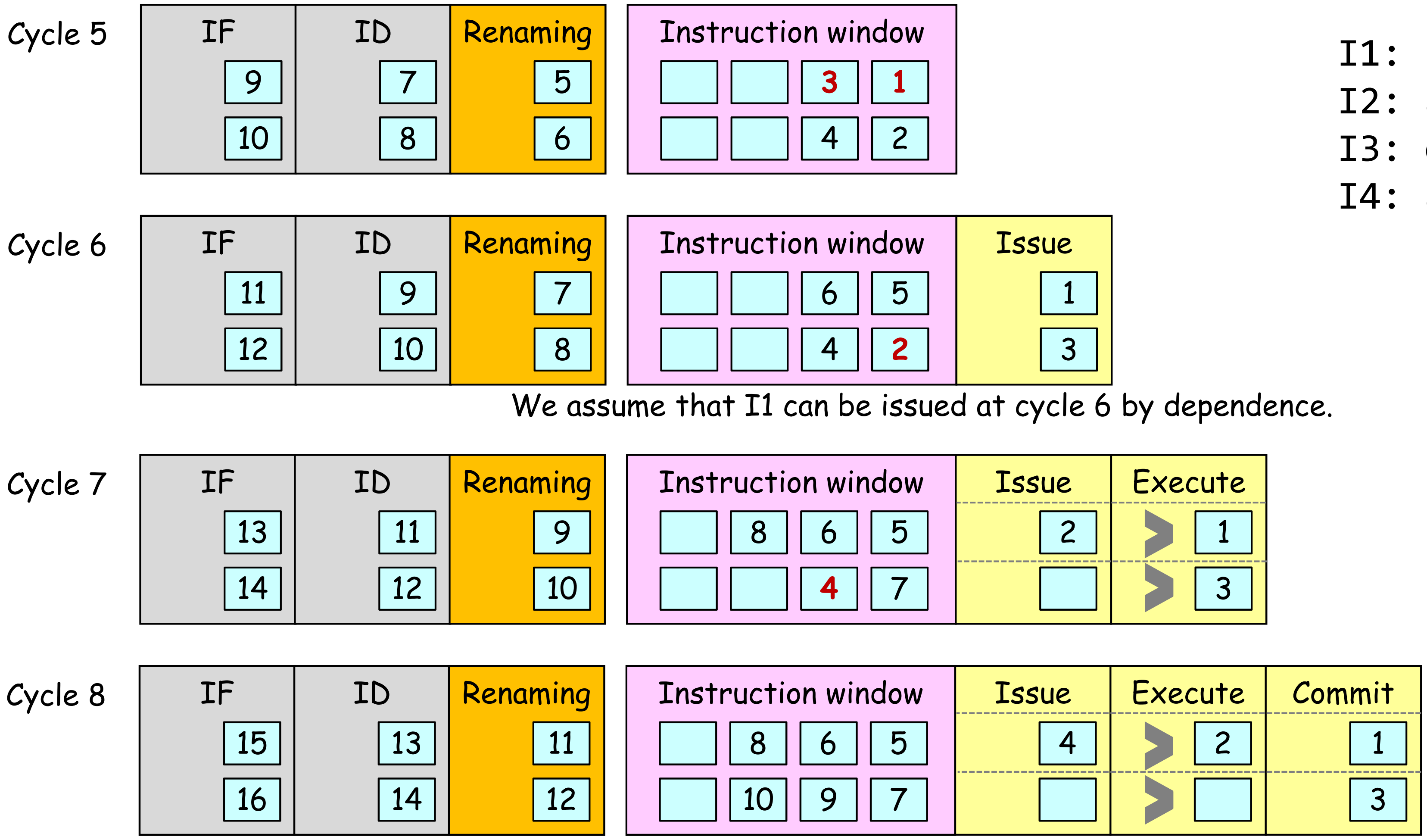
```
I1: sub p9,p1,p2
I2: add p10,p9,p3
I3: or  p11,p4,p5
I4: and p12,p10,p11
```



Data flow graph

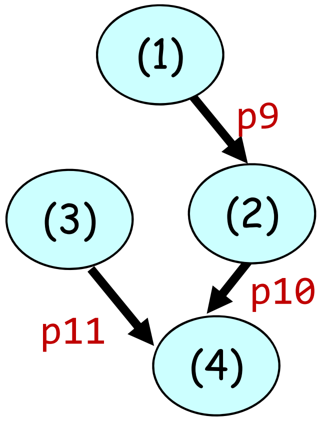
The key idea for OoO execution (2/3)

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



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

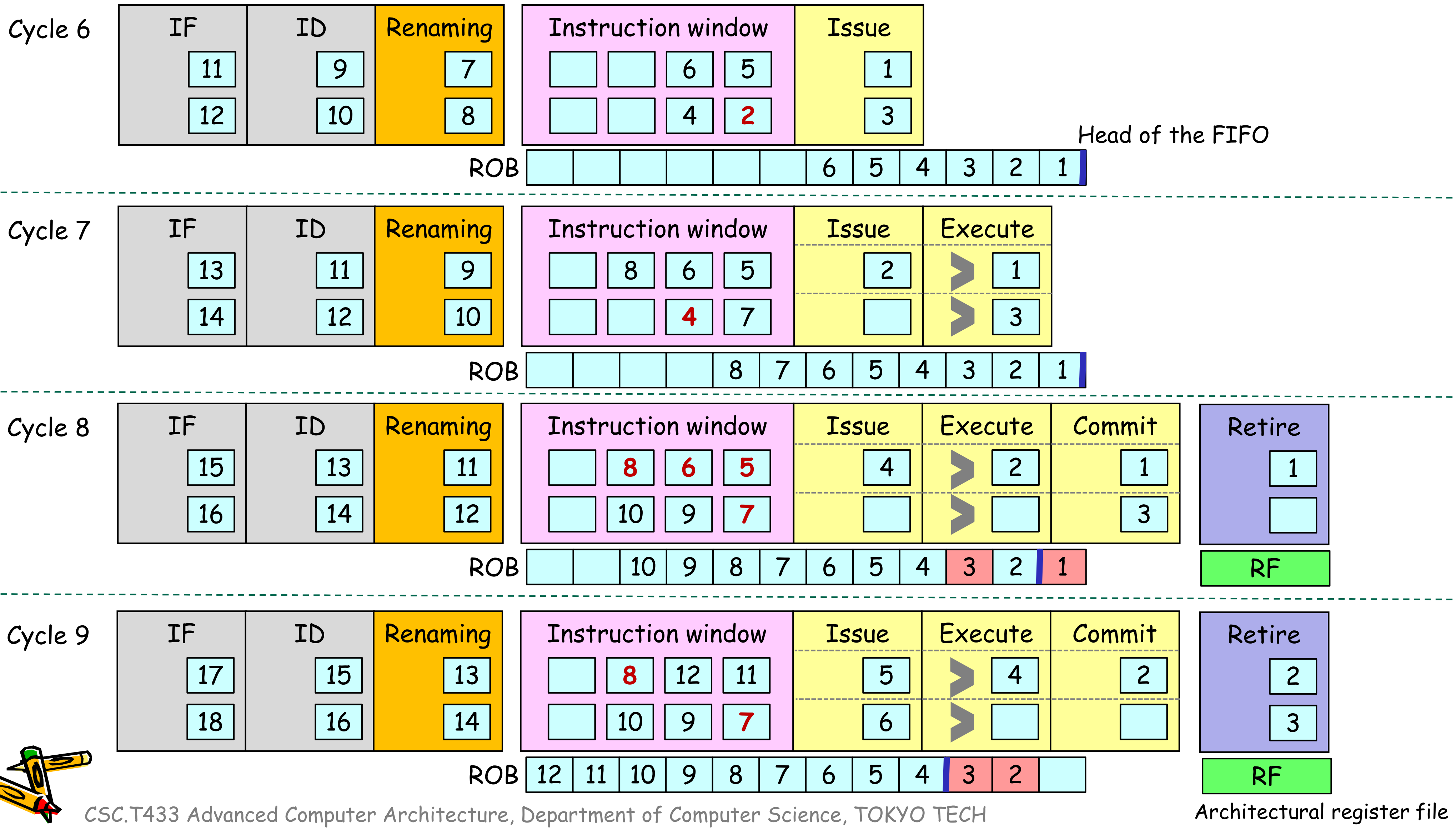
```
I1: sub p9,p1,p2
I2: add p10,p9,p3
I3: or  p11,p4,p5
I4: and p12,p10,p11
```



Data flow graph

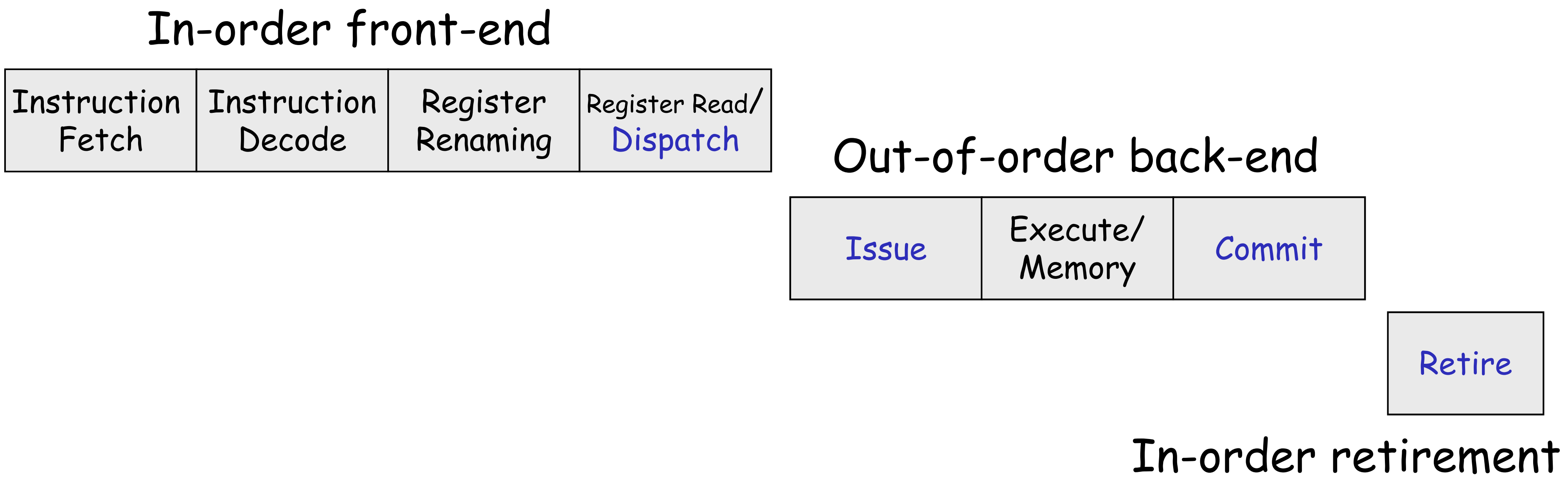
The key idea for OoO execution (3/3)

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



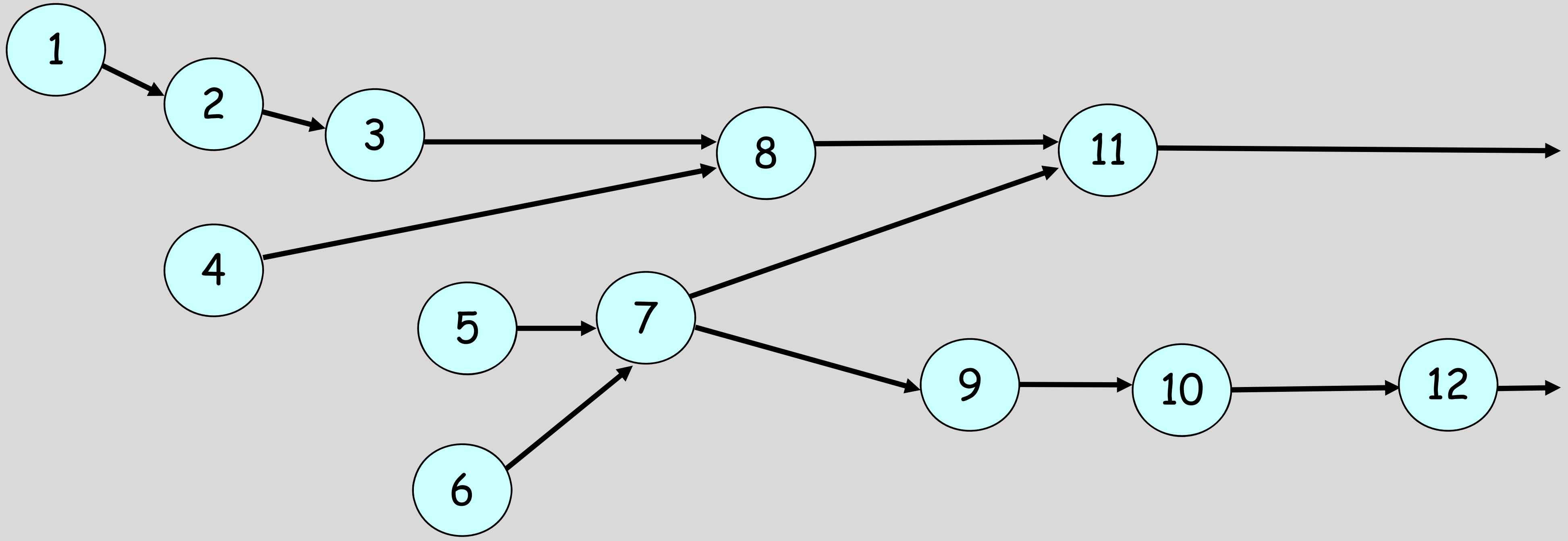
Instruction pipeline of OoO execution processor

- 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 result is written back to register file (architectural register file) using a logical register number.

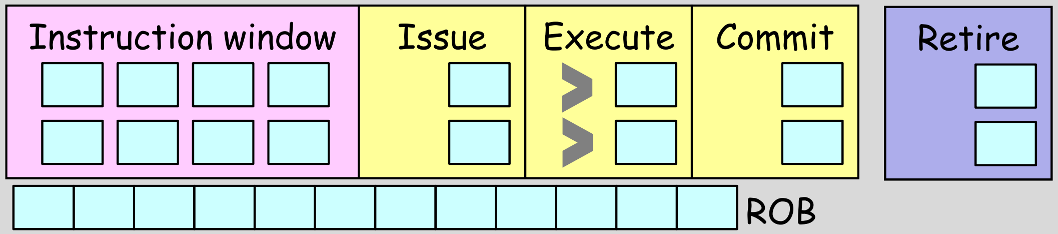


Exercise: OoO execution

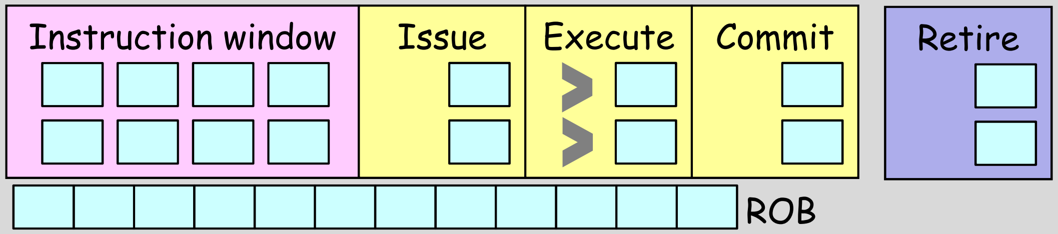
- Draw the cycle by cycle processing behavior of these 12 instructions
  - wakeup
  - select



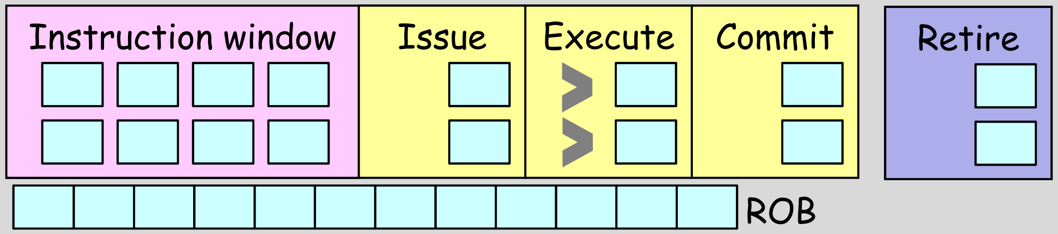
Cycle 1



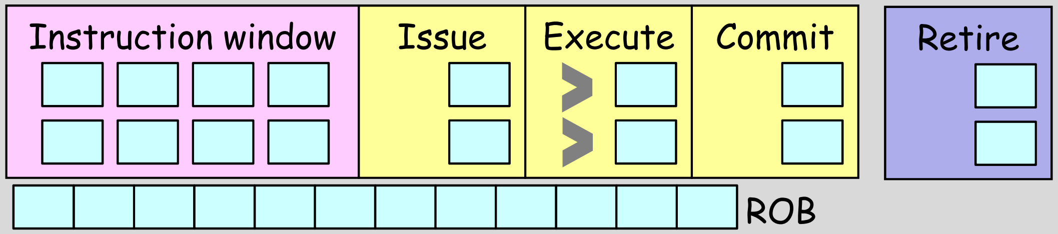
Cycle 2



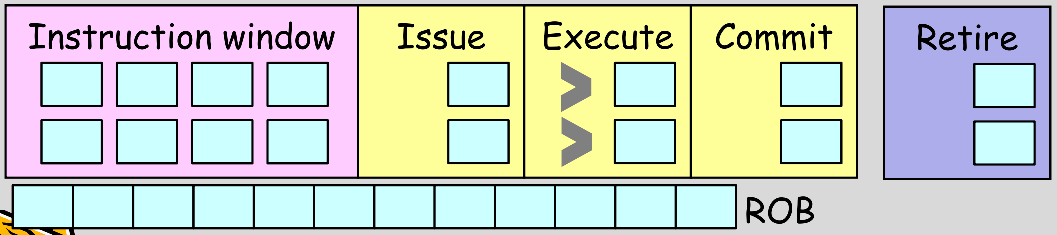
Cycle 3



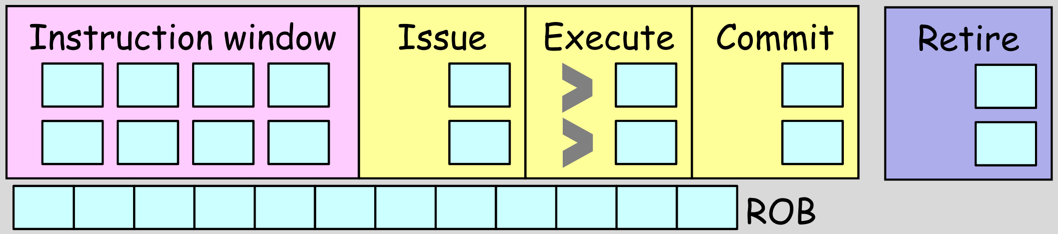
Cycle 4



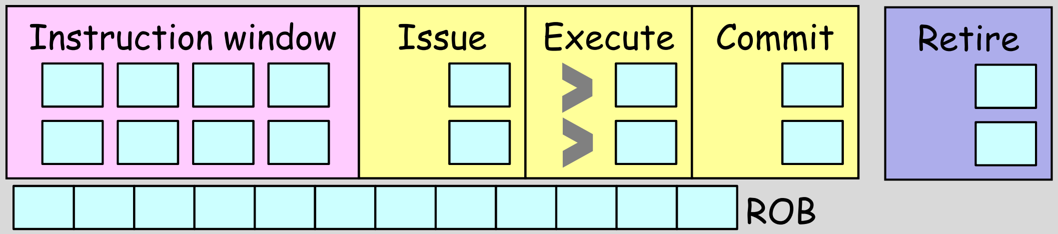
Cycle 5



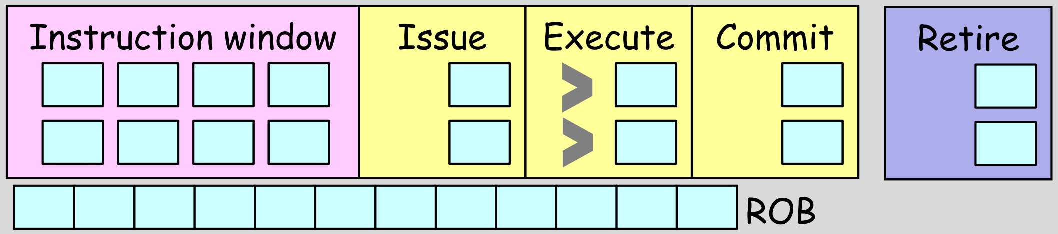
Cycle 6



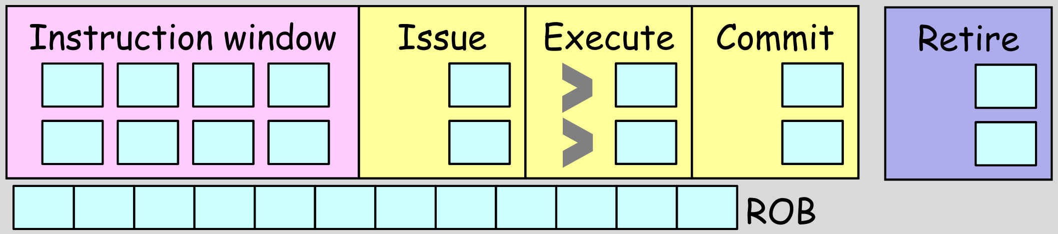
Cycle 7



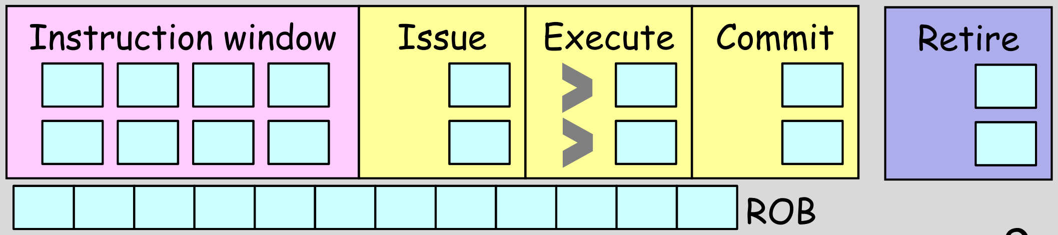
Cycle 8



Cycle 9

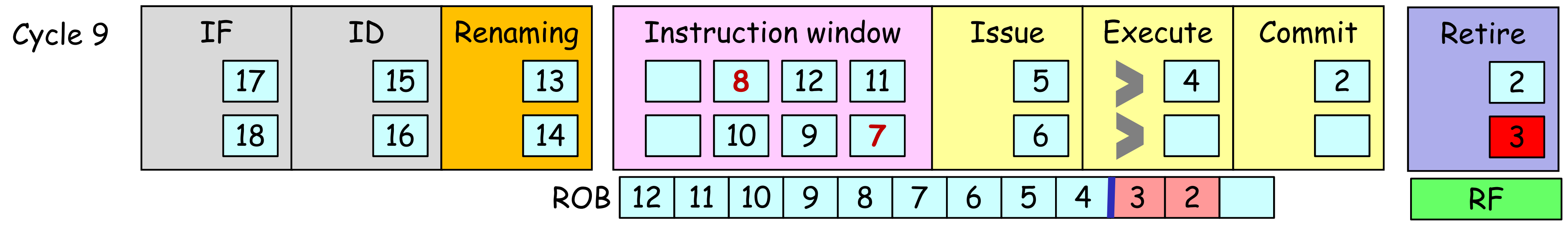


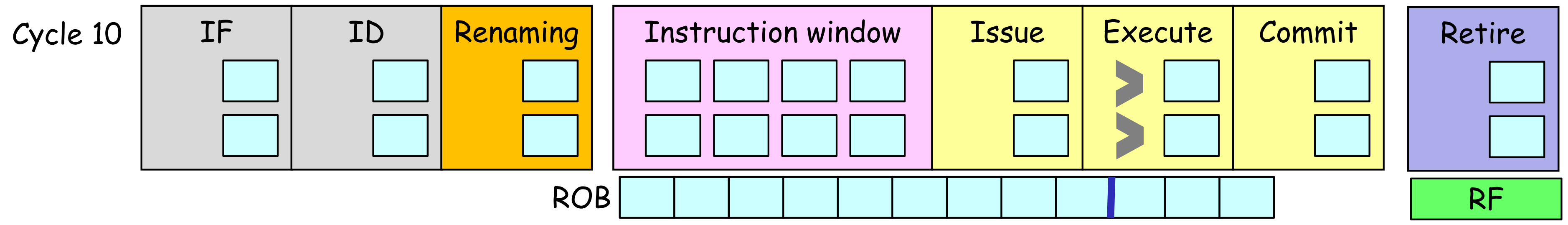
Cycle 10



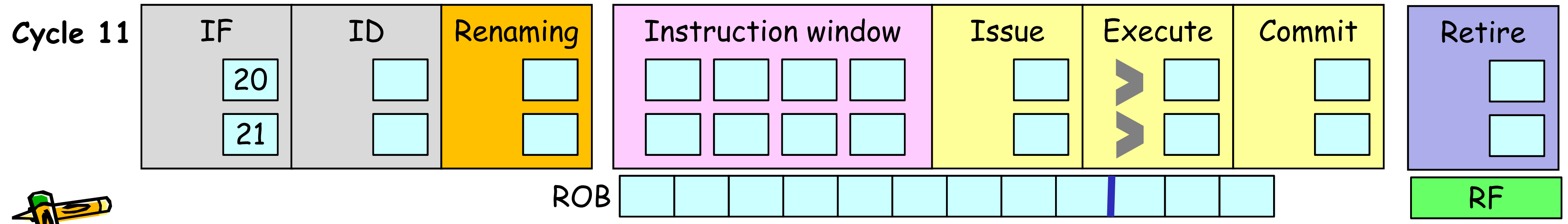
Prediction miss and recovery

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





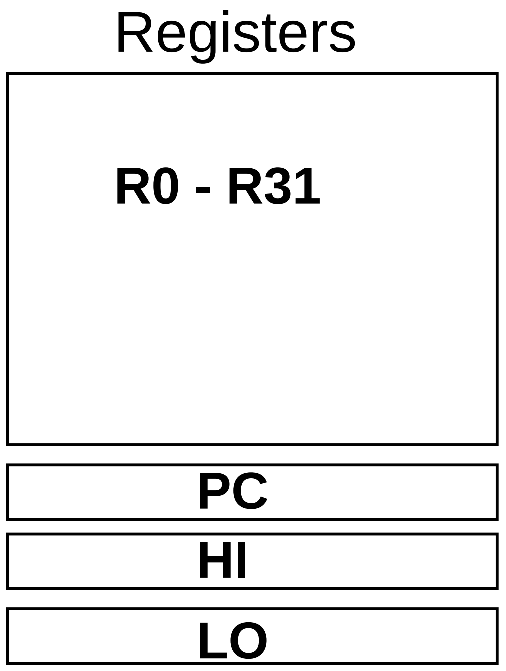
Recovery by flushing instructions on the wrong path (may takes several cycles)



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

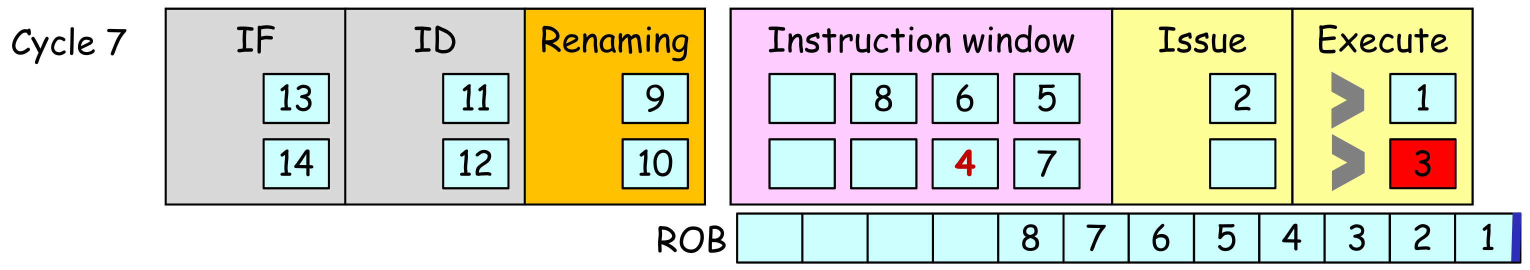


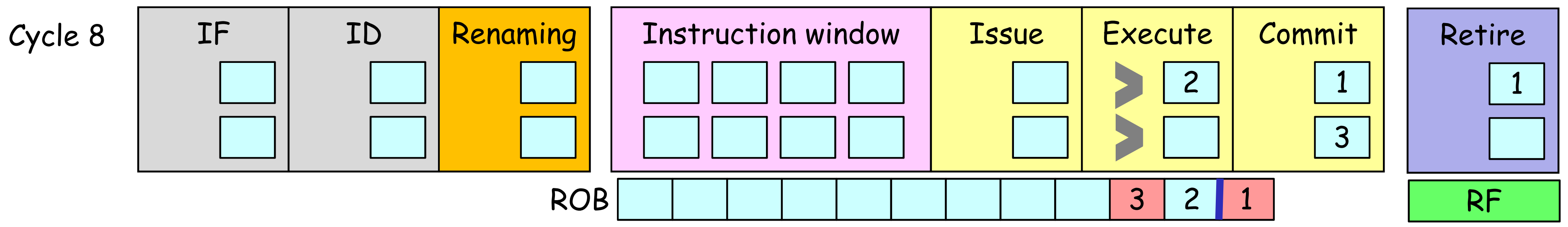
3 Instruction Formats: all 32 bits wide

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

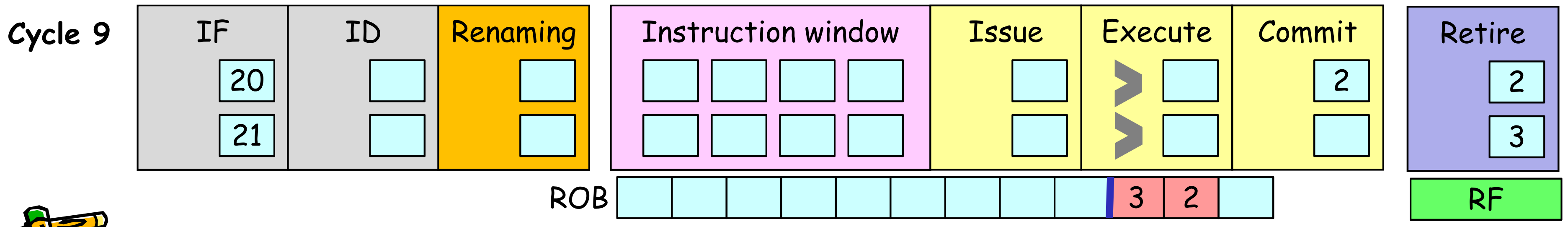
Branch prediction miss and aggressive recovery

- Instruction 3 is a miss predicted branch and its target insn is 20
- Register file (and PC) has the architecture state after insn 3 is executed
- When insn 3 is executed, recover by flushing instructions after insn 3 and restart





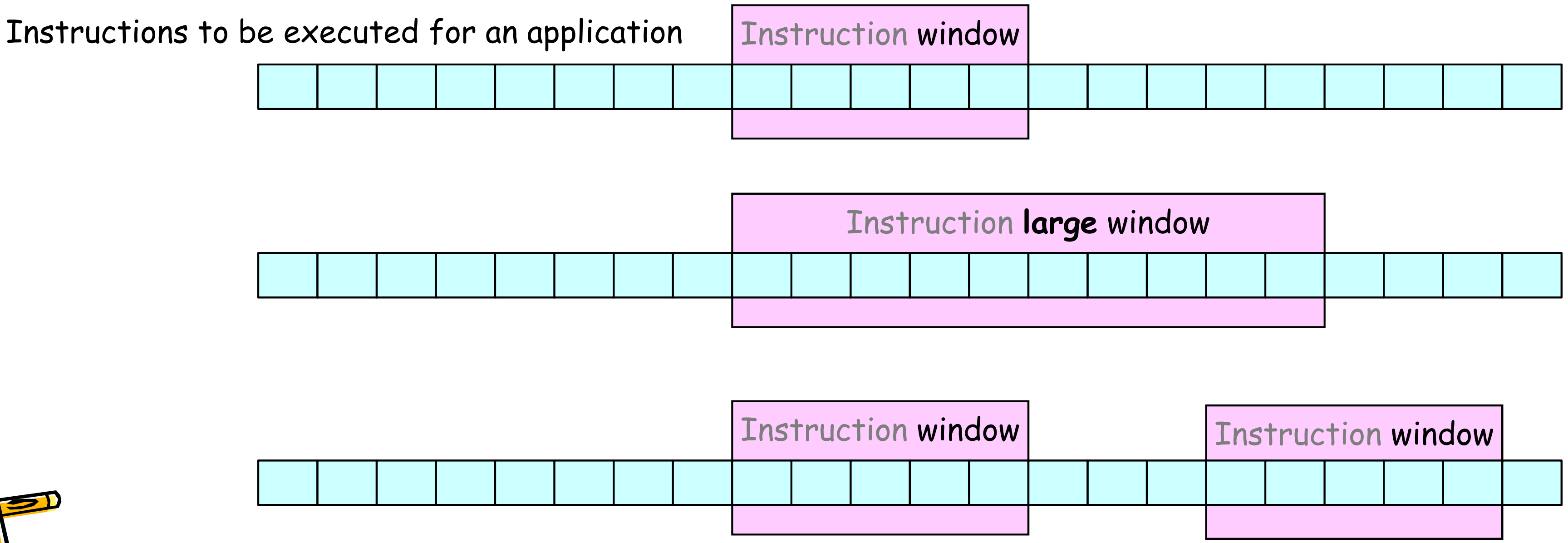
Recovery by flushing instructions on the wrong path (may takes several cycles)



Restart by fetching instructions using the correct PC

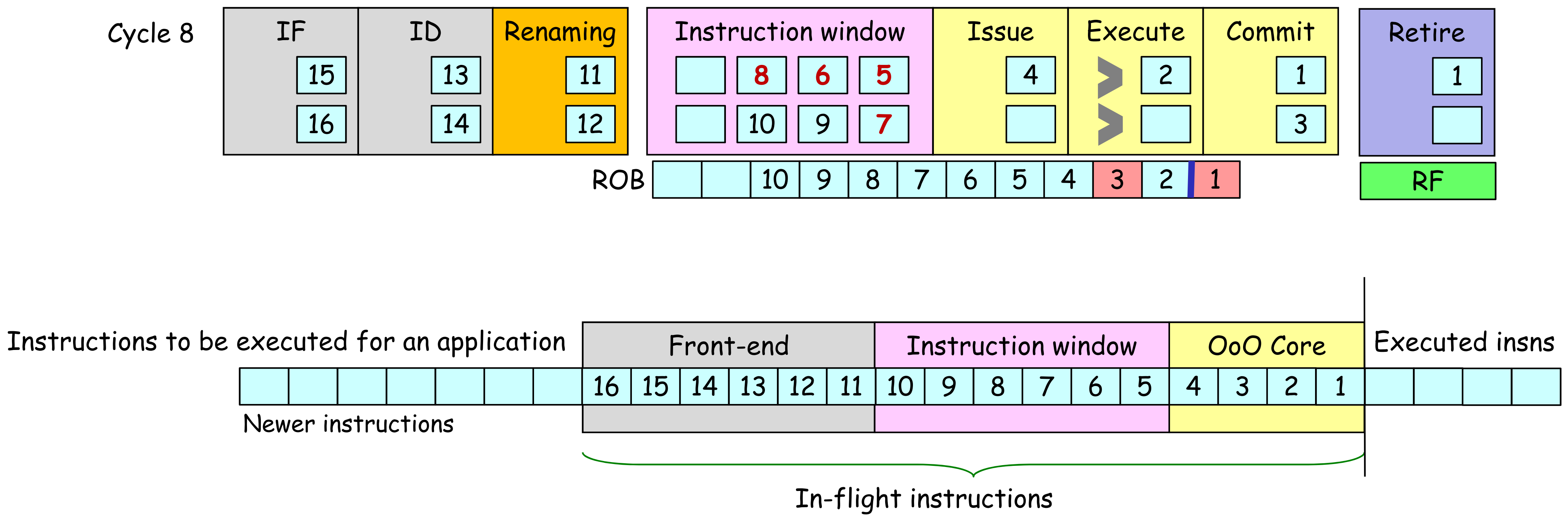
Aside: What is a window?

- 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)


Instructions to be executed for an application
Instruction window
Instruction **large** window
Instruction window
Instruction window

Register dataflow

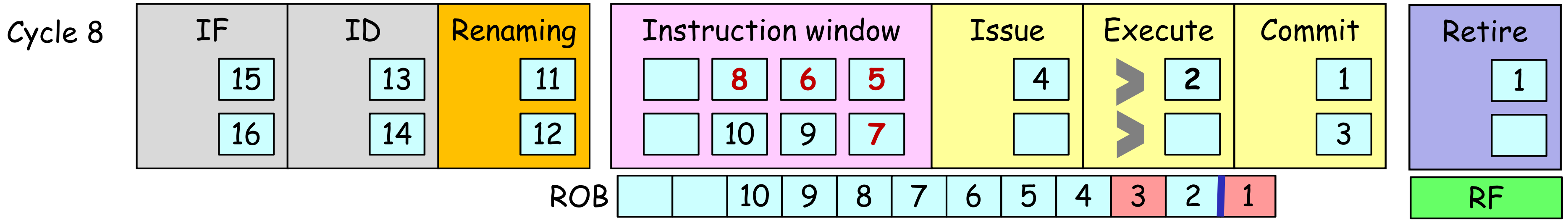
- In-flight instructions are ones processing in a processor

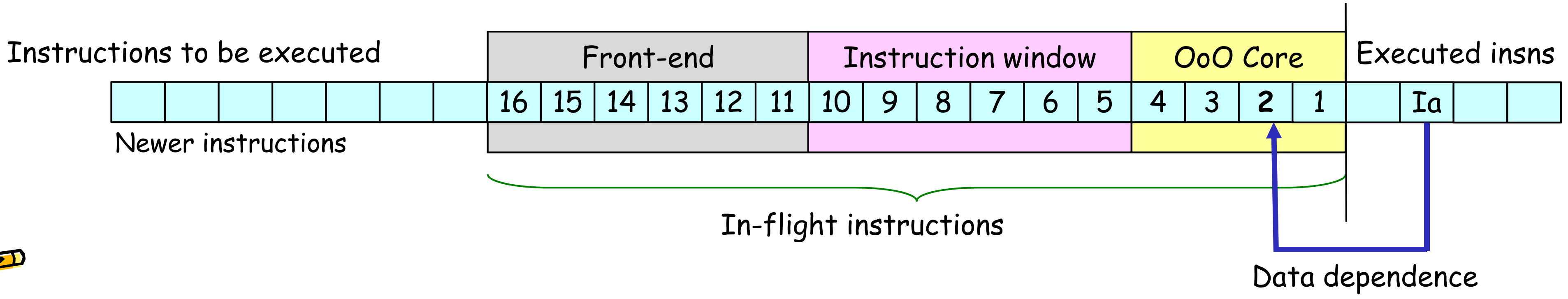


Case 1: Register dataflow from a far previous instn

- One source operand of insn I2 is from a retired instruction Ia.
- Because Ia is retired, the destination register has no renamed tag. The tag of a source register can not be renamed at renaming stage, still having a logical register tag $3.
- Where does the operand $3 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
```



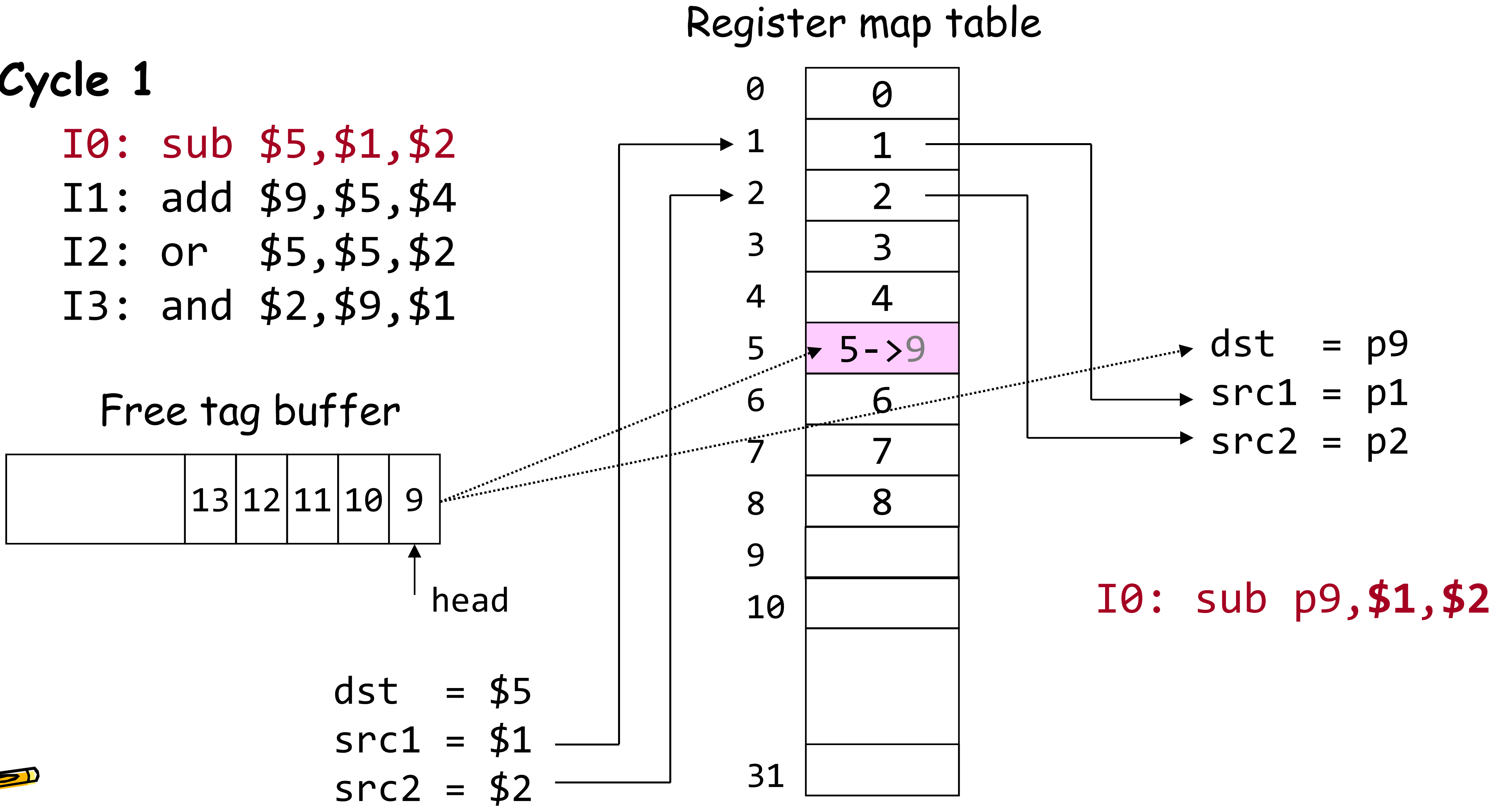


Register renaming again

- A processor remembers a set of renamed logical registers.
- If $1 and $2 are not renamed in in-flight instructions, it uses $1 and $2 instead of p1 and p2.

Cycle 1

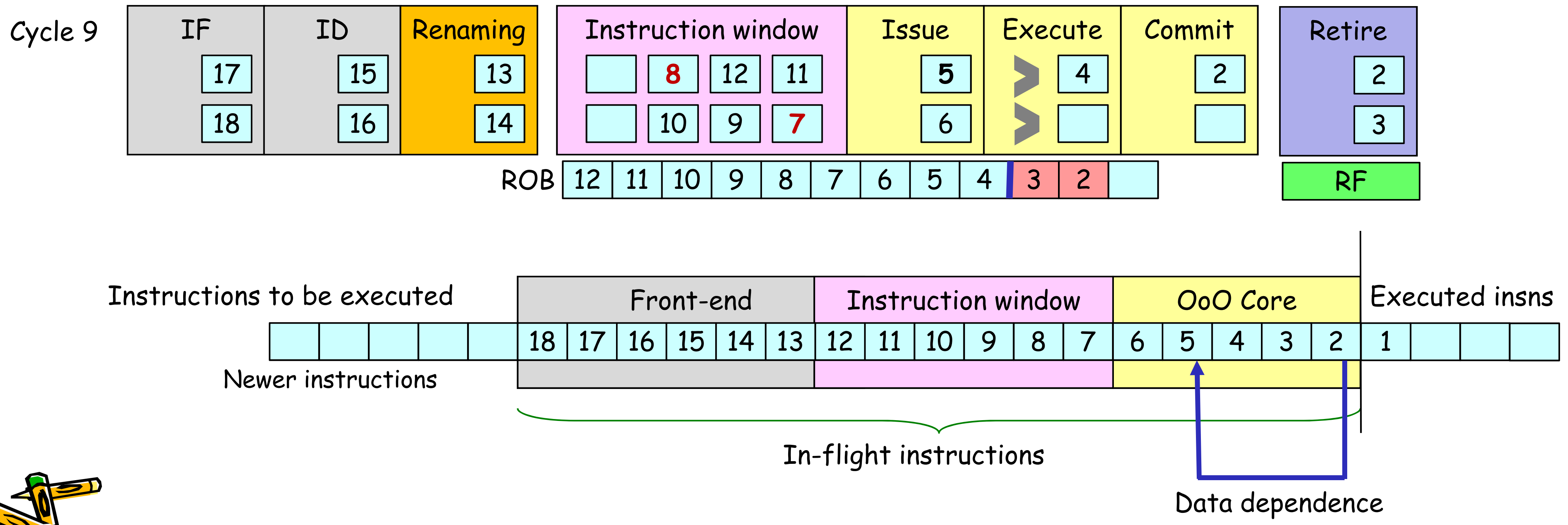
```
I0: sub $5,$1,$2
I1: add $9,$5,$4
I2: or  $5,$5,$2
I3: and $2,$9,$1
```



I0: sub p9,**$1**,**$2**

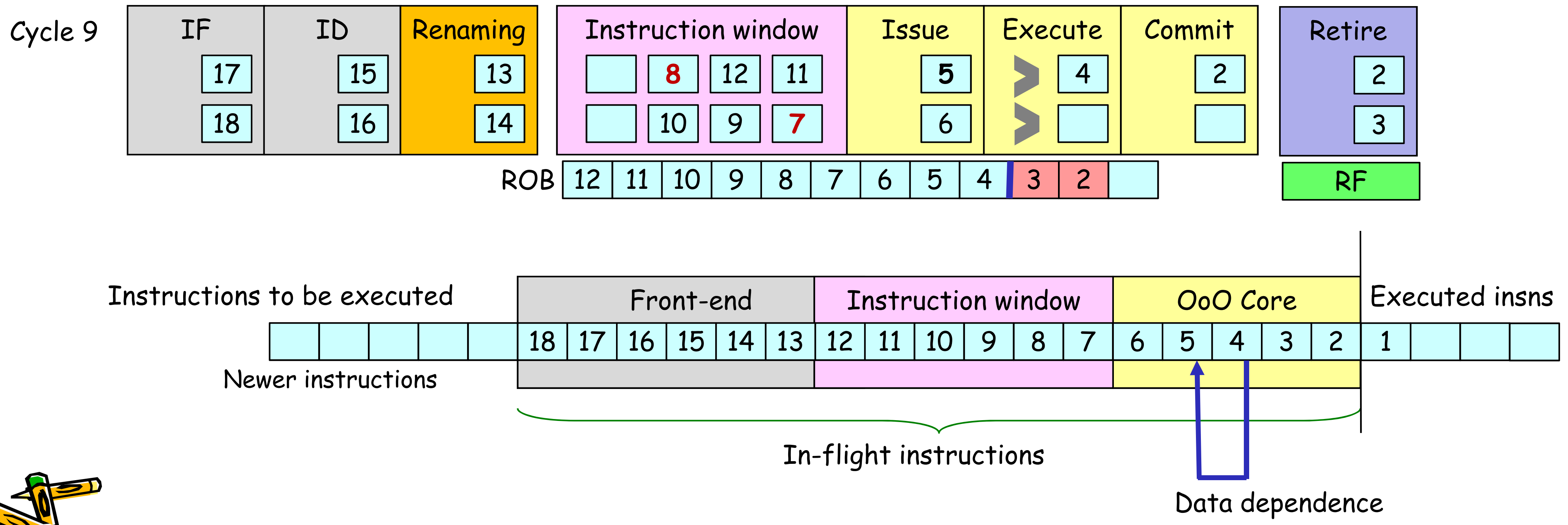
Case 2: Register dataflow from ROB

- Assume that one source operand of insn I5 is from I2 which is not retired. The operand is generated a few clock cycles (sometimes, tens of cycles) earlier.
- Because I2 is not retired, RF does not have the operand. I2 is committed, so the operand is stored in ROB.
- Where does the operand comes from?

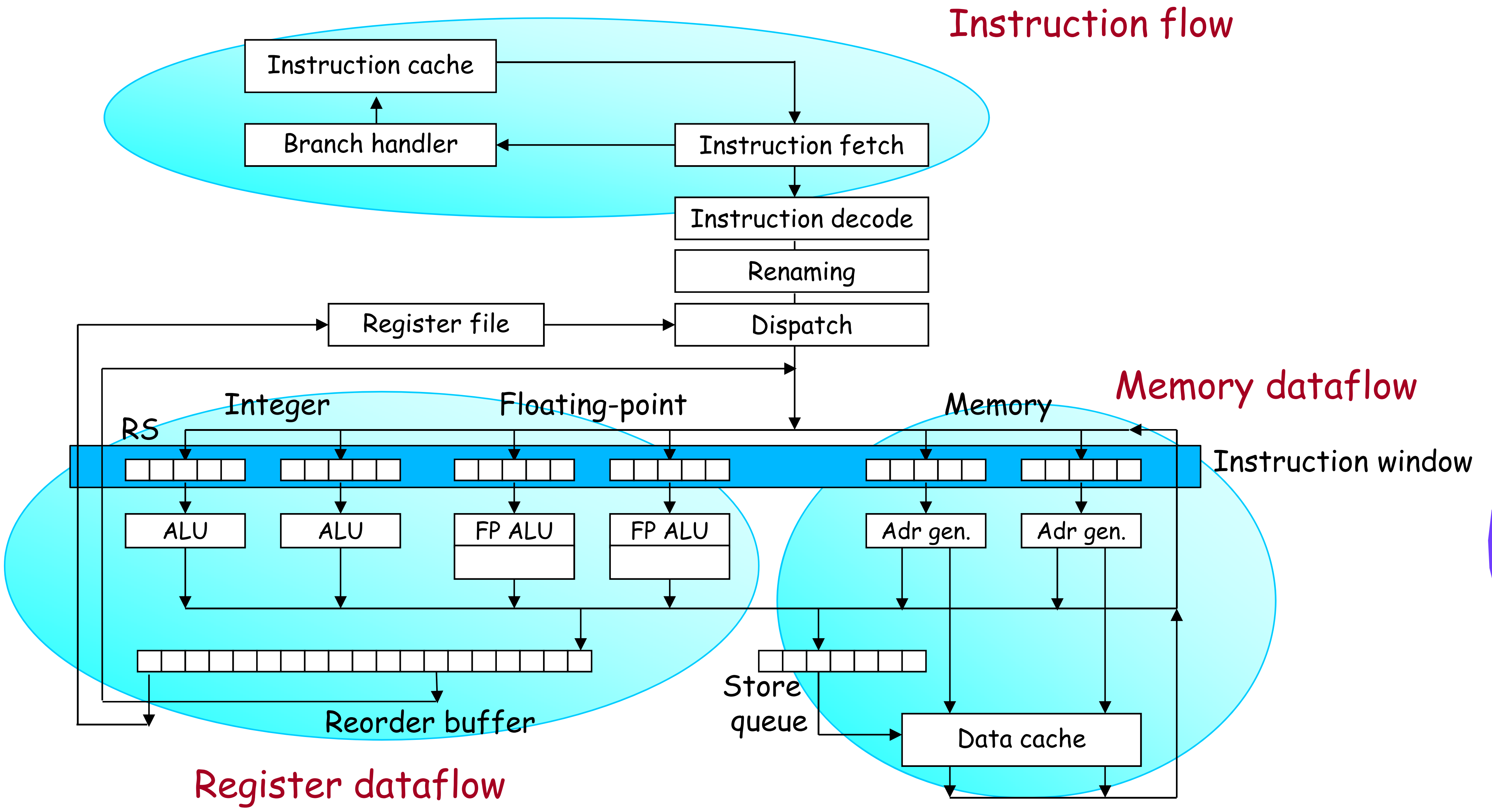


Case 3: Register dataflow from ALUs

- Assume that one source operand of insn I5 is from I4 which is not retired. The operand is generated in the previous clock cycle.
- Because I2 is not retired, RF does not have the operand.
  Because I2 is not committed, ROB does not have the operand.
- Where does the operand comes from?



Datapath of OoO execution processor



Reservation station (RS)

Pollack's Rule

- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity". Complexity in this context means processor logic, i.e. its area.

WIKIPEDIA