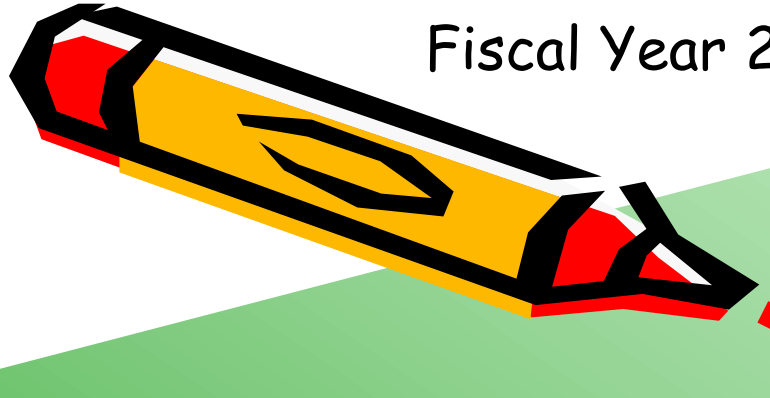


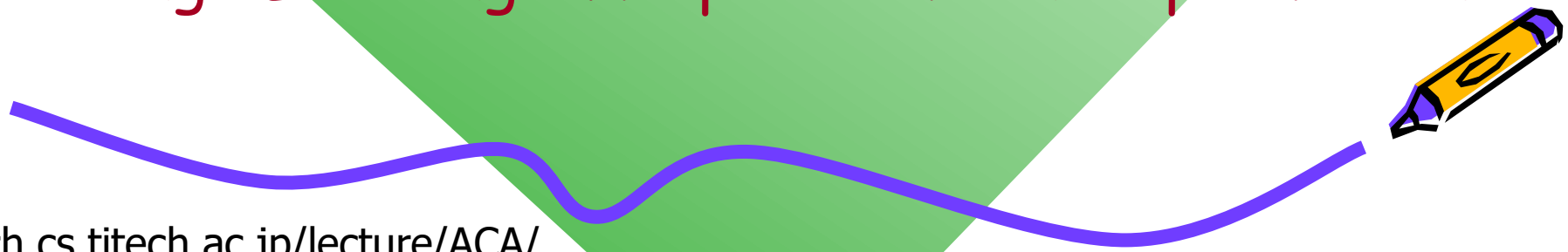
Fiscal Year 2023

Ver. 2024-01-15a

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.W834, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

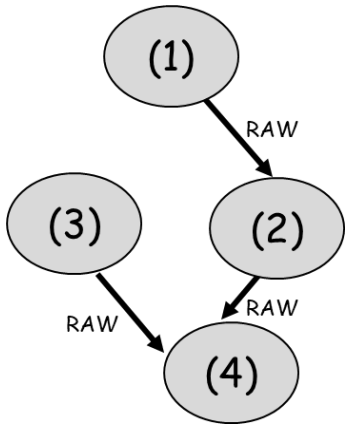
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Exploiting Instruction Level parallelism (ILP)

- A superscalar has to handle some flows efficiently to exploit ILP
  - Control flow (control dependence)
    - To execute ***n*** instructions per clock cycle, the processor has to fetch at least ***n*** instructions per cycle.
    - The main obstacles are branch instruction (BNE)
    - Prediction
    - Another obstacle is instruction cache
  - Register data flow (data dependence)
    - **Out-of-order execution**
      - Register renaming
      - Dynamic scheduling
  - Memory data flow
    - Out-of-order execution
    - Another obstacle is data cache

```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

```
(3) lw  x4, 4(x7)
(1) add x5,x1,x2
(2) add x9,x5,x3
(4) add x8,x9,x4
```



Hardware register renaming (last lecture)

- Logical registers (architectural registers) which are ones defined by ISA
  - x0, x1, … x31
- Physical registers
  - Assuming plenty of registers are available, p0, p1, p2, …
- A processor renames (converts) each logical register to a unique physical register dynamically in the renaming stage

Typical instruction pipeline of scalar processor

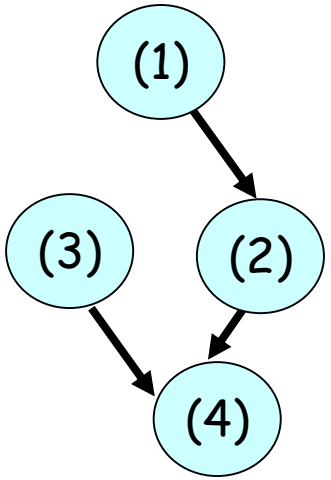
Typical instruction pipeline of high-performance superscalar processor

dequeue & allocate

collect & enqueue

Out-of-order execution (OoO execution)

- In in-order execution model, all instructions are executed in the order that they appear as (1), (2), (3), (4) ...
  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)
  - A key design philosophy behind OoO execution to extract ILP by executing instructions as quickly as possible.
    - Scoreboarding (CDC6600 in 1964)
    - Tomasulo algorithm
      (IBM System/360 Model 91 in 1967)

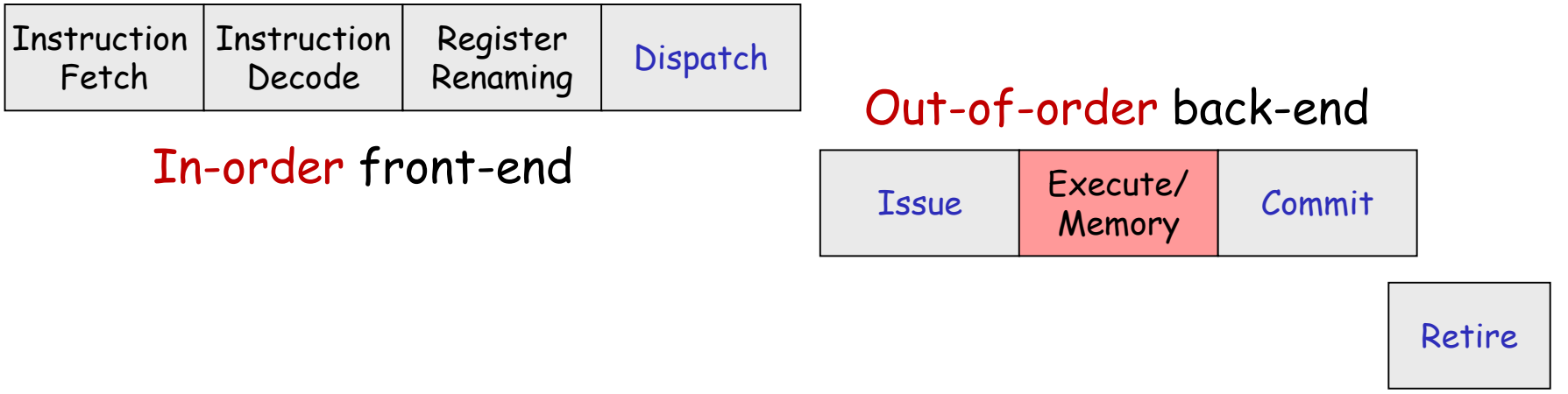



```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

Data flow graph

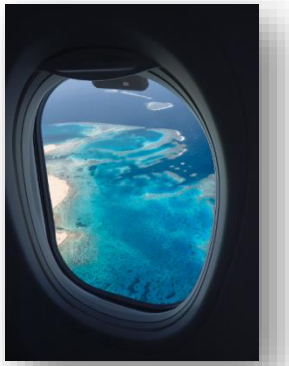
Instruction pipeline of OoO execution processor

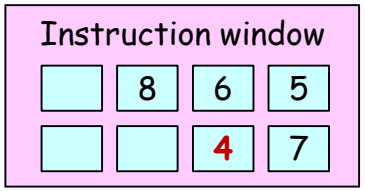
- Allocating instructions to instruction window is called dispatch
- Issue or fire wakes up instructions and their executions begin
- In commit stage, *the computed values are written back to ROB (reorder buffer)*
- The last stage is called retire or graduate. The completed consecutive instructions can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

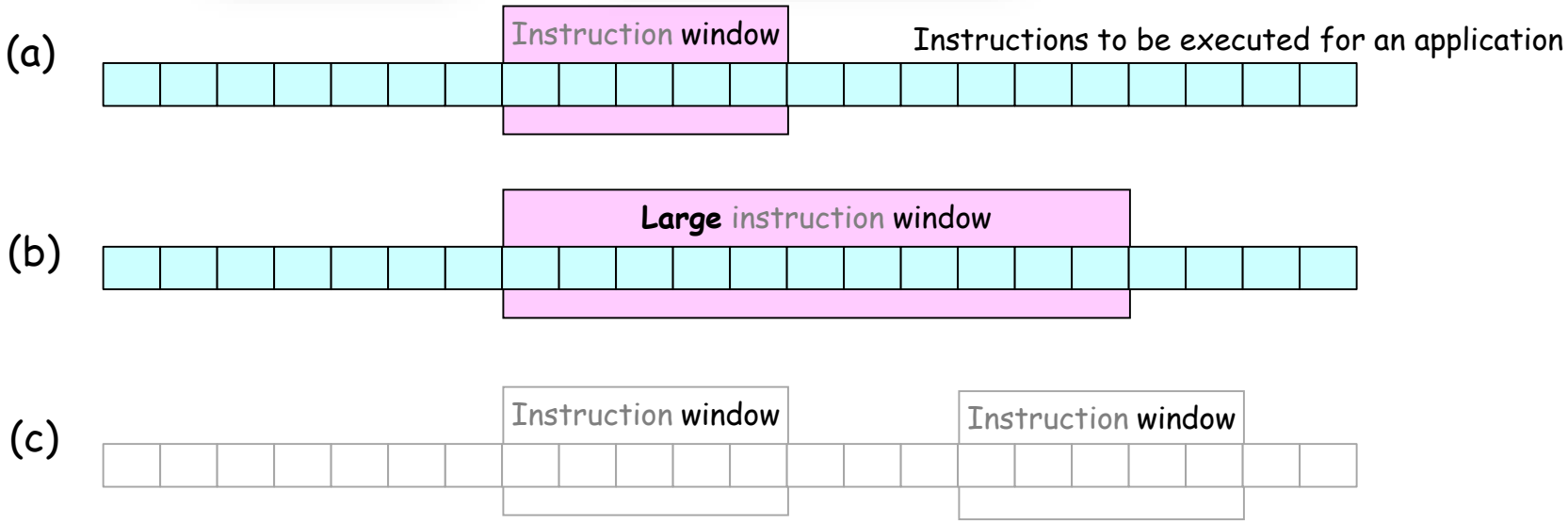


Aside: What is a window?

- 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)

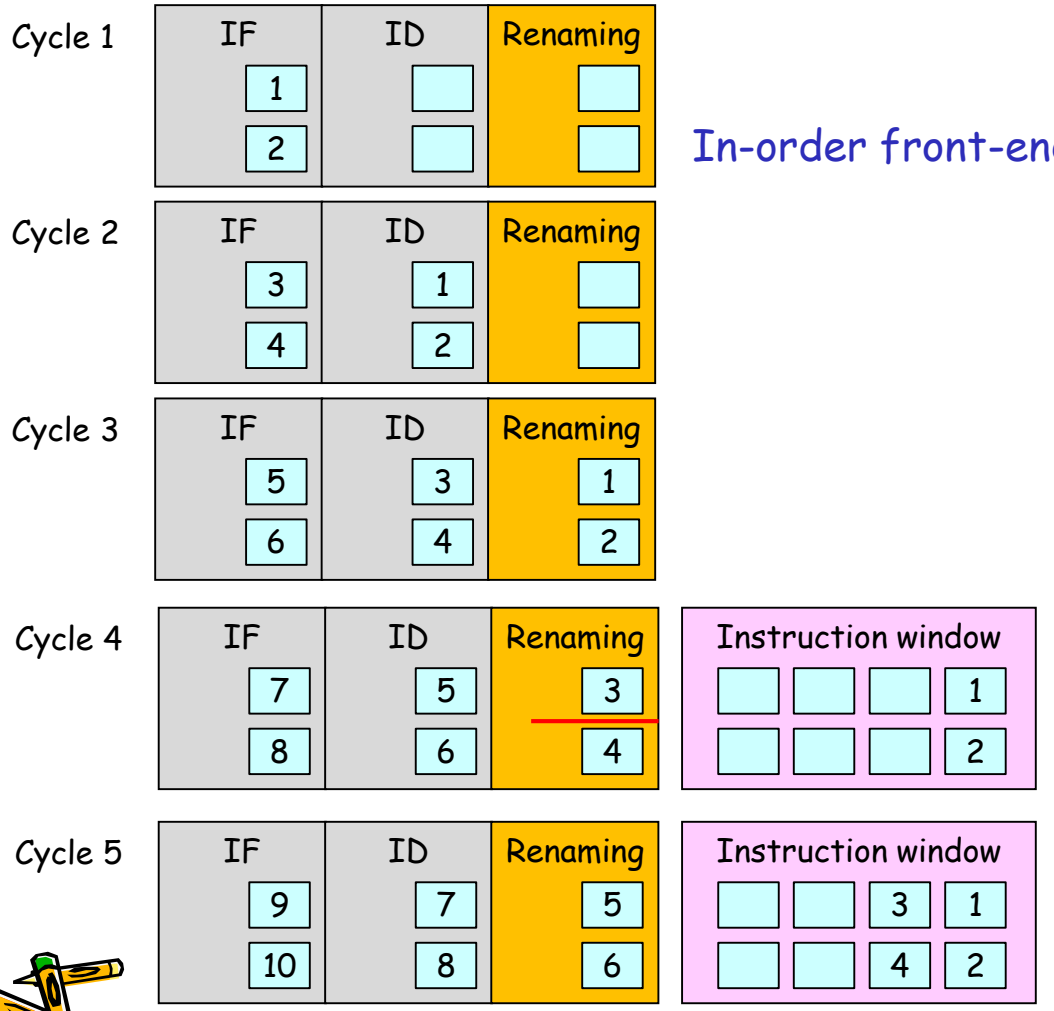




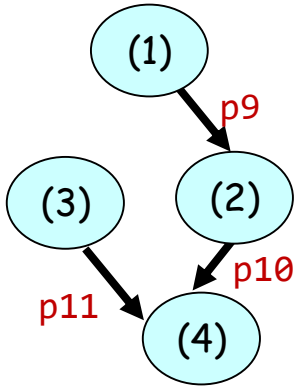


The key idea for OoO execution (1/3)

- 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

assume that instructions cannot exit the instruction window until cycle 5

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)

Cycle 5

IF	ID	Renaming	Instruction window
9	7	5	3 1
10	8	6	4 2

Cycle 6

IF	ID	Renaming	Instruction window	Issue
11	9	7	6 5	1
12	10	8	4 2	3

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

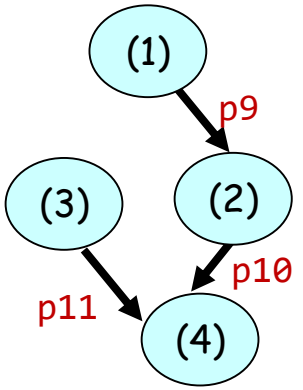
Cycle 7

IF	ID	Renaming	Instruction window	Issue	Execute
13	11	9	8 6 5	2	> 1
14	12	10	4 7		> 3

Cycle 8

IF	ID	Renaming	Instruction window	Issue	Execute	Commit
15	13	11	8 6 5	4	> 2	1
16	14	12	10 9 7		>	3

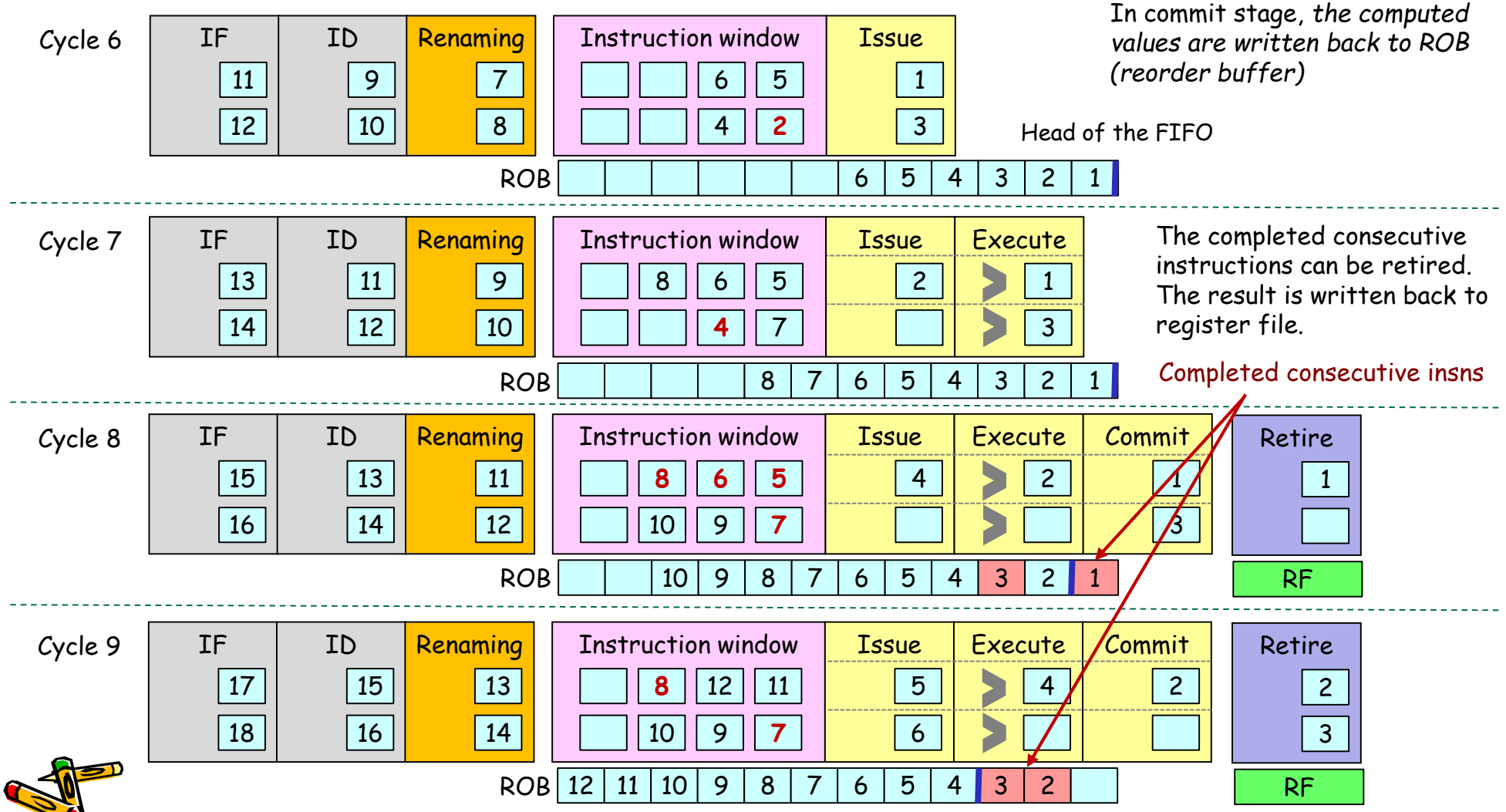
```
I1: sub p9,p1,p2
I2: add p10,p9,p3
I3: or  p11,p4,p5
I4: and p12,p10,p11
```



Data flow graph

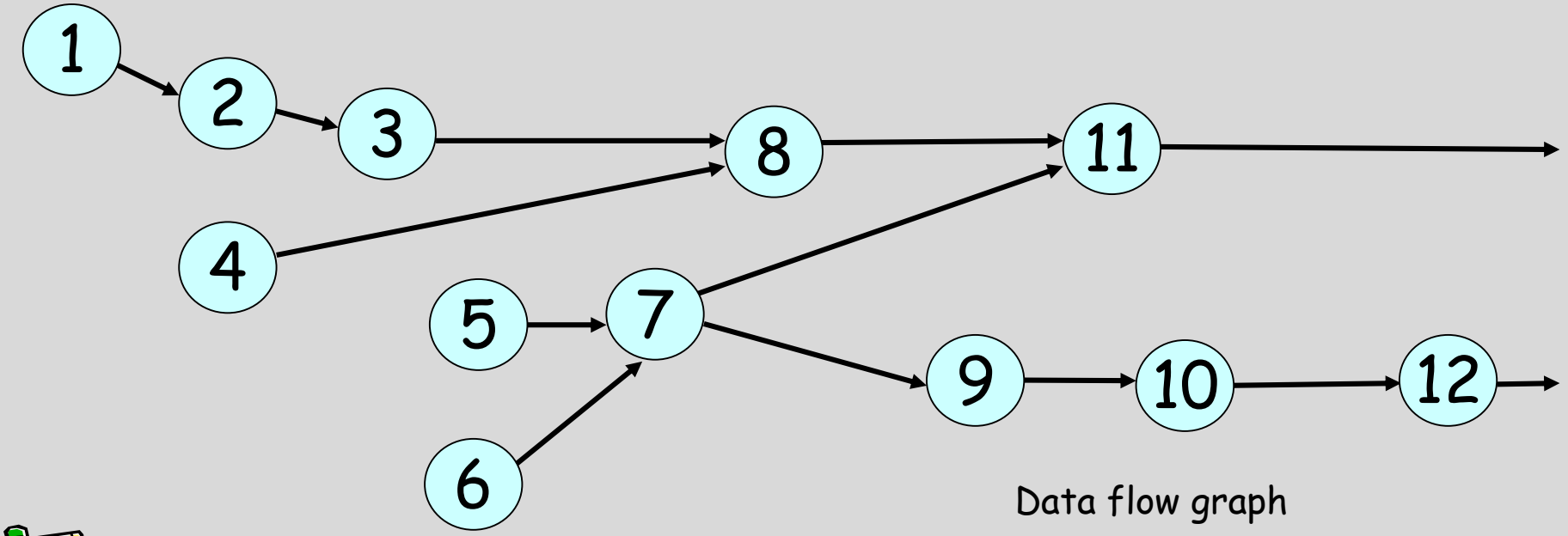
The key idea for OoO execution (3/3)

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

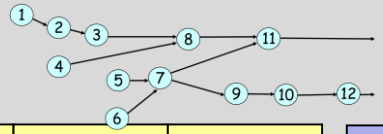


Exercise 1

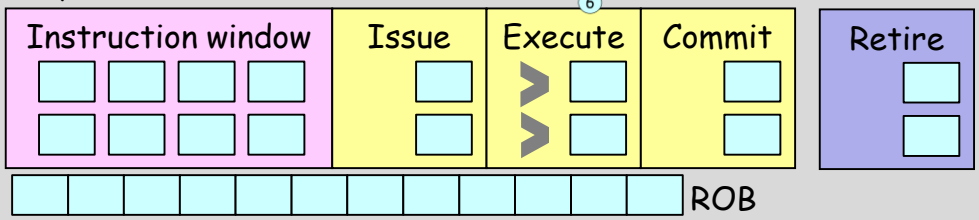
- OoO execution
- Fill out the cycle by cycle processing behavior of these 12 instructions
  - wakeup
  - select



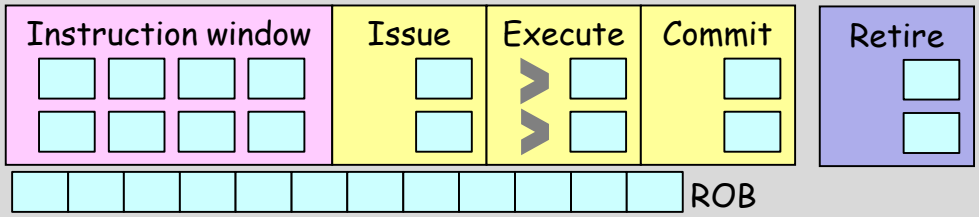
Data flow graph



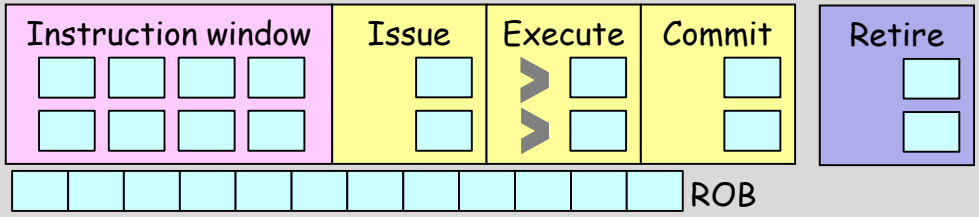
Cycle 1



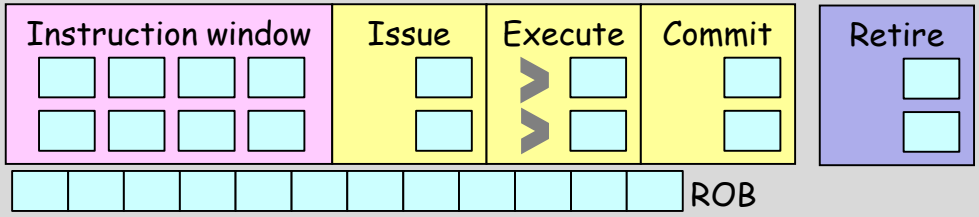
Cycle 2



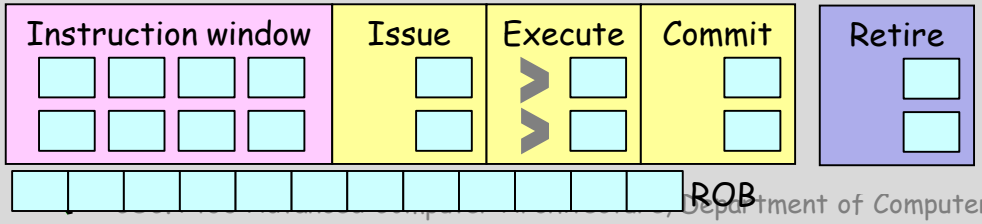
Cycle 3

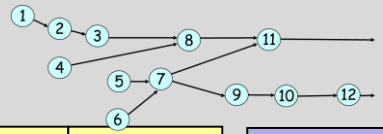


Cycle 4

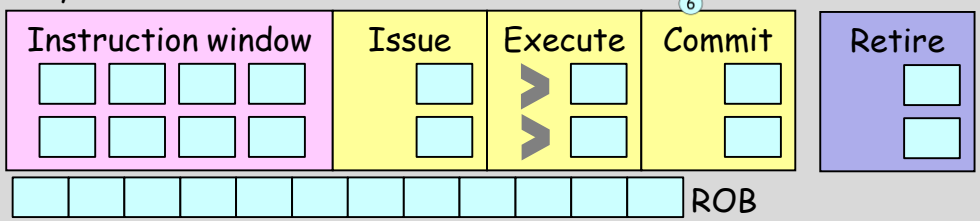


Cycle 5

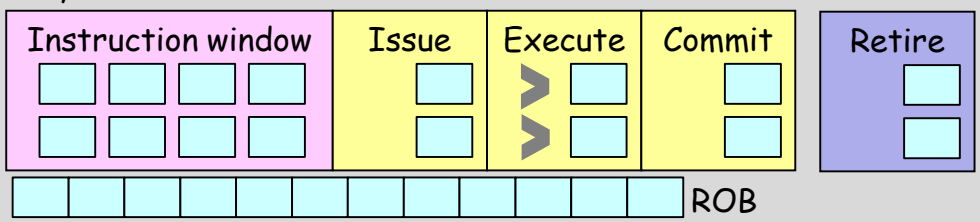




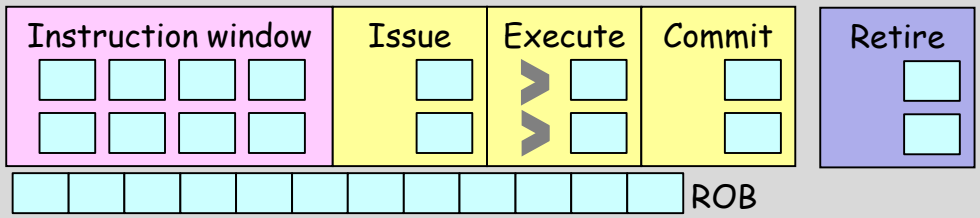
Cycle 6



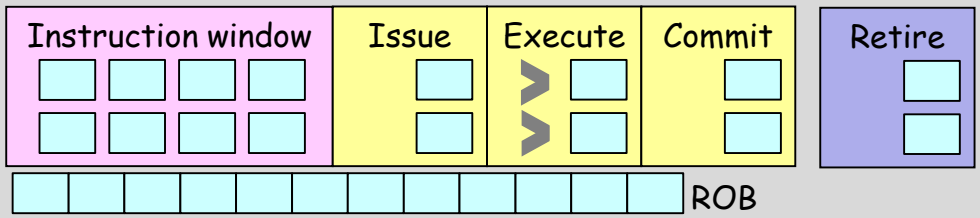
Cycle 7



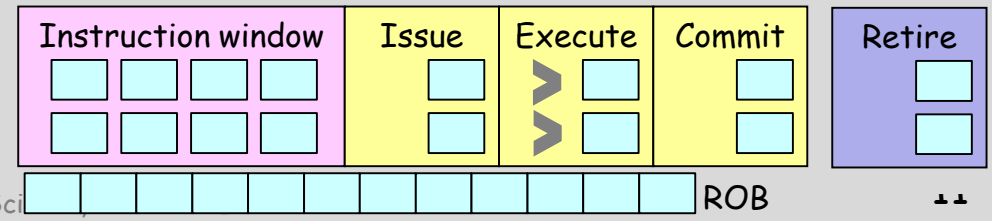
Cycle 8



Cycle 9

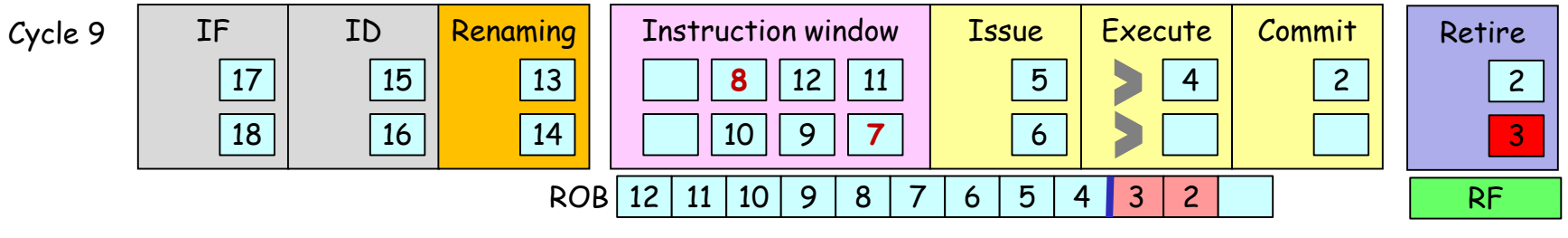


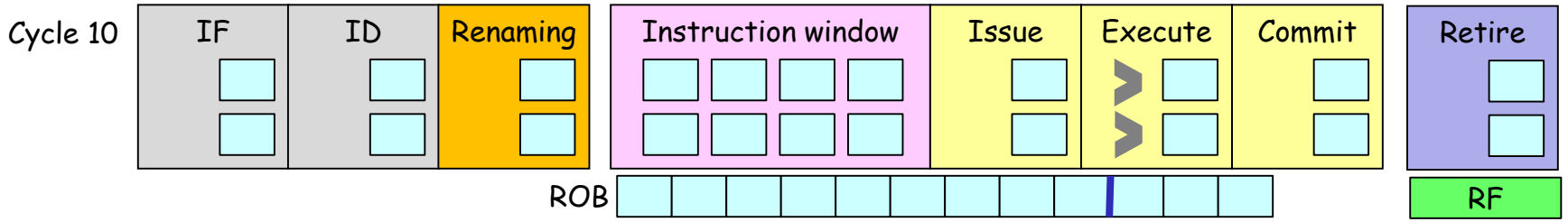
Cycle 10



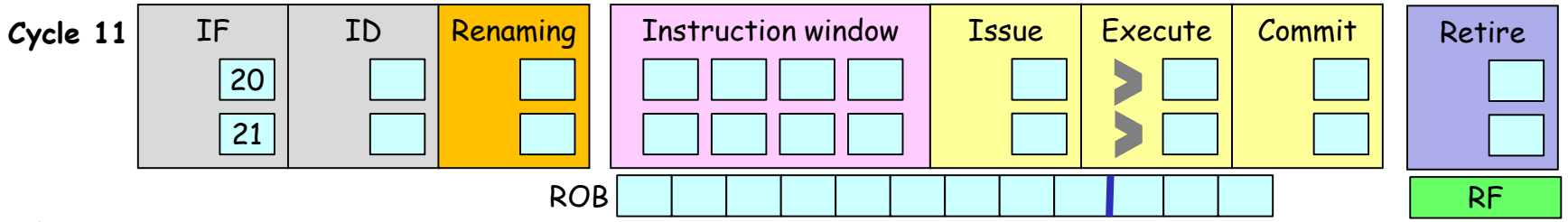
Prediction miss and recovery

- 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





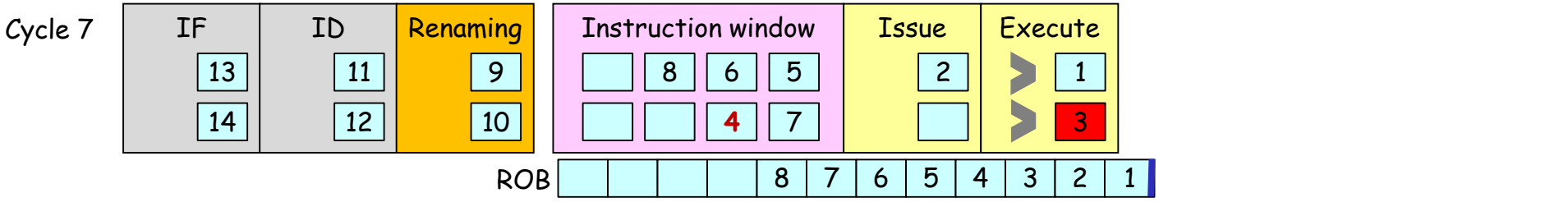
Recovery by flushing instructions on the wrong path (may take several cycles)

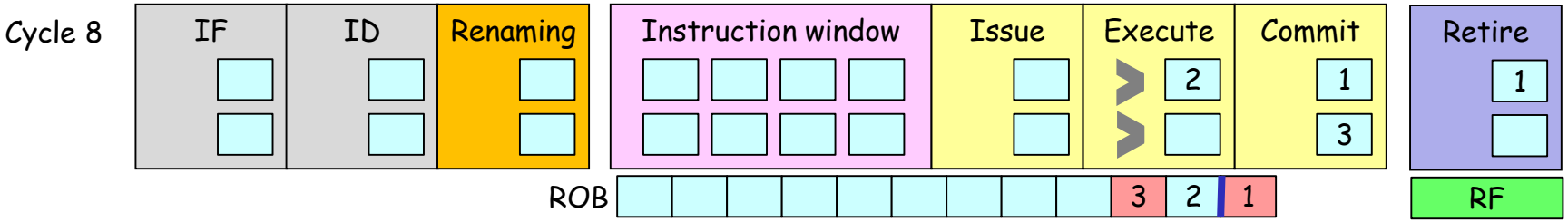


Restart by fetching instructions using the correct PC

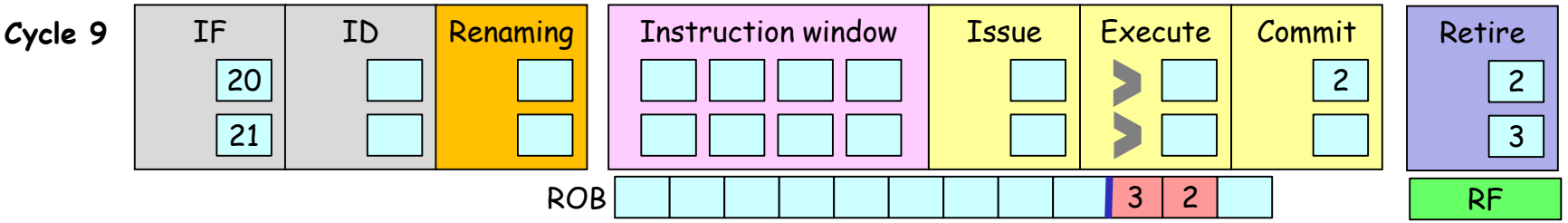
Branch prediction miss and aggressive recovery

- 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





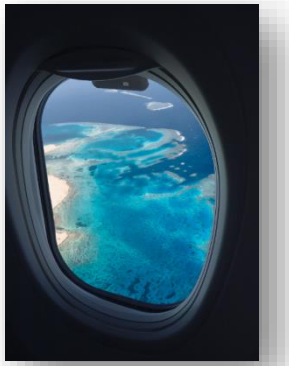
Recovery by flushing instructions on the wrong path (may takes several cycles)

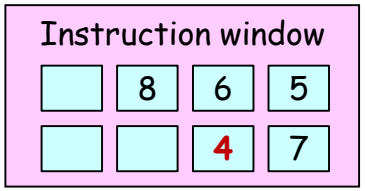


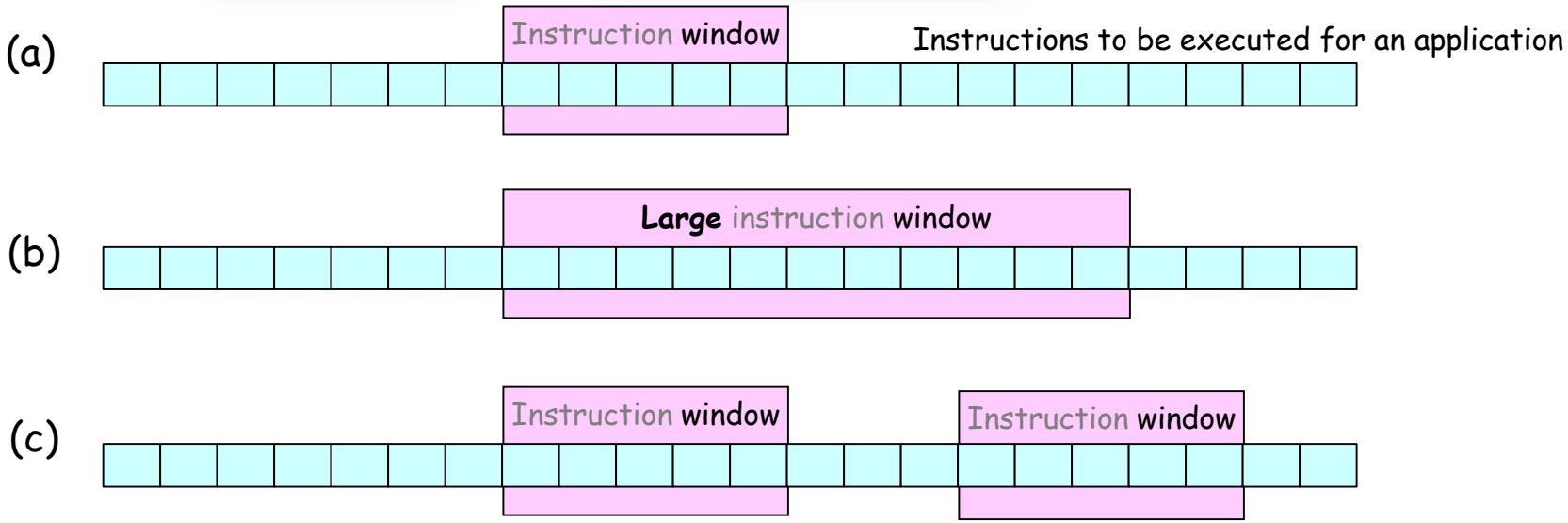
Restart by fetching instructions using the correct PC

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)

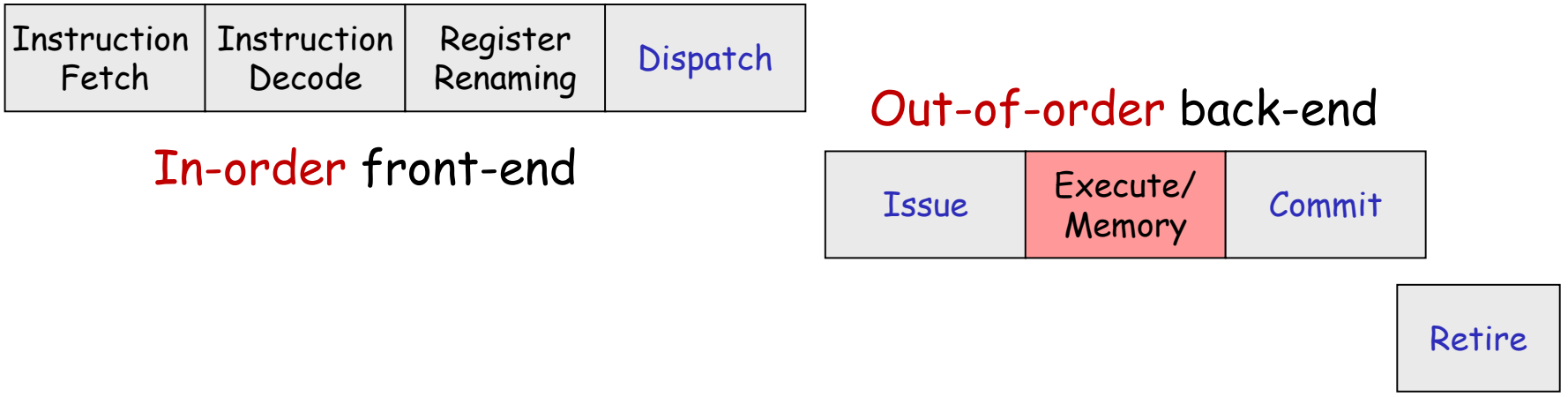






Instruction pipeline of OoO execution processor

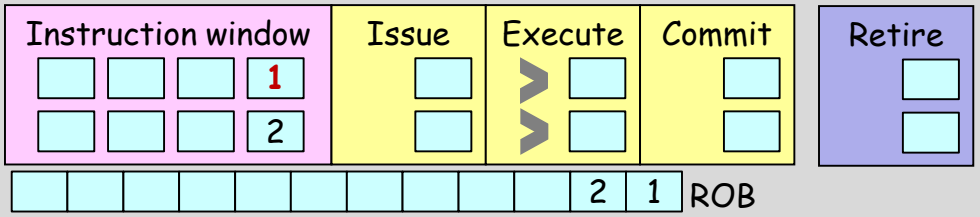
- Allocating instructions to instruction window is called dispatch
- Issue or fire wakes up instructions and their executions begin
- In commit stage, *the computed values are written back to ROB (reorder buffer)*
- The last stage is called retire or graduate. The completed consecutive instructions can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



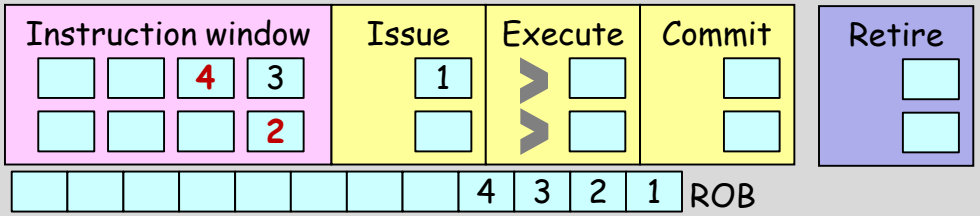
Recommended Reading

- Clockhands: Rename-free Instruction Set Architecture for Out-of-order Processors
  - Toru Koizumi (NITech), Ryota Shioya, Shu Sugita, Taichi Amano, Yuya Degawa, Junichiro Kadomoto, Hidetsugu Irie, Shuichi Sakai (U.Tokyo)
  - 56th IEEE/ACM International Symposium on Microarchitecture (MICRO'23)
- A quote:
  "Out-of-order superscalar processors are currently the only architecture that speeds up irregular programs, but they suffer from poor power efficiency. To tackle this issue, we focused on how to specify register operands. Specifying operands by register names, as conventional RISC does, requires register renaming, resulting in poor power efficiency and preventing an increase in the front-end width. In contrast, a recently proposed architecture called STRAIGHT specifies operands by inter-instruction distance, thereby eliminating register renaming. However, STRAIGHT has strong constraints on instruction placement, which generally results in a large increase in the number of instructions.
- We propose Clockhands, a novel instruction set architecture that has multiple register groups and specifies a value as "the value written in this register group k times before." Clockhands does not require register renaming as in STRAIGHT. In contrast, Clockhands has much looser constraints on instruction placement than STRAIGHT, allowing programs to be written with almost the same number of instructions as Conventional RISC. We implemented a cycle-accurate simulator, FPGA implementation, and first-step compiler for Clockhands and evaluated benchmarks including SPEC CPU. On a machine with an eight-fetch width, the evaluation results showed that Clockhands consumes 7.4% less energy than RISC while having performance comparable to RISC. This energy reduction increases significantly to 24.4% when simulating a futuristic up-scaled processor with a 16-fetch width, which shows that Clockhands enables a wider front-end."

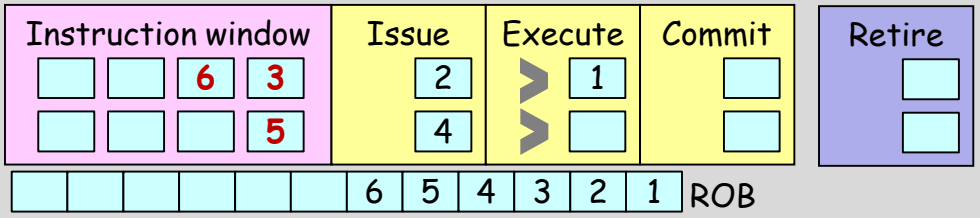
Cycle 1



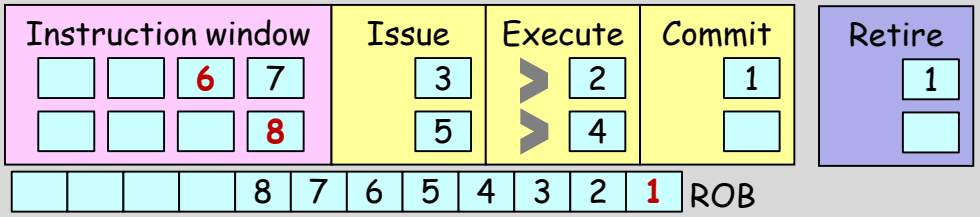
Cycle 2



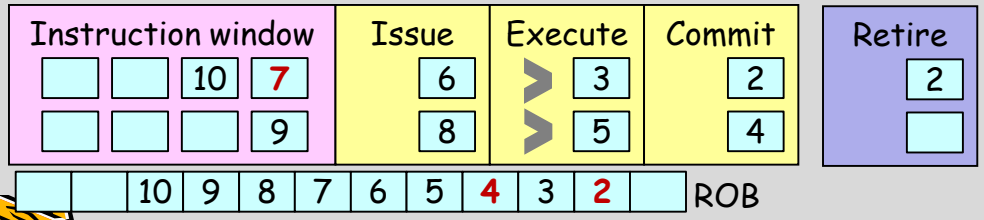
Cycle 3



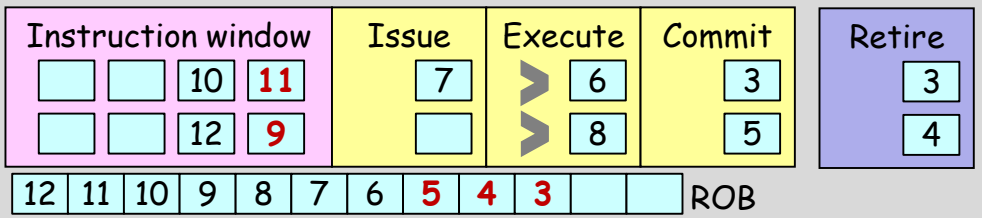
Cycle 4



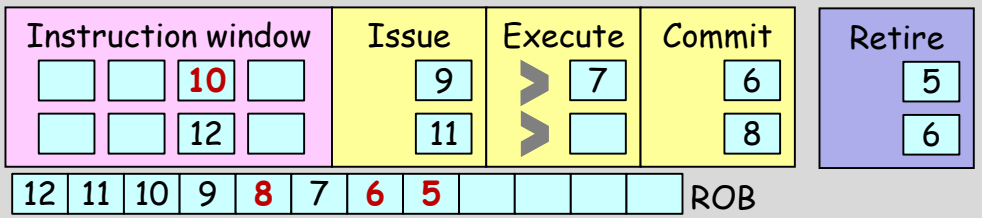
Cycle 5



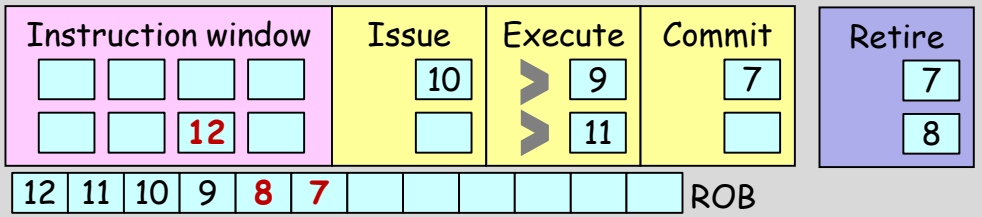
Cycle 6



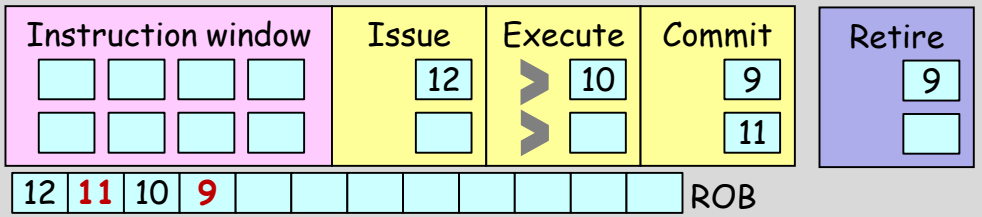
Cycle 7



Cycle 8



Cycle 9



Cycle 10