Fiscal Year 2023

Ver. 2024-01-18a

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

Advanced Computer Architecture

9. Instruction Level Parallelism: Out-of-order Execution and Multithreading

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

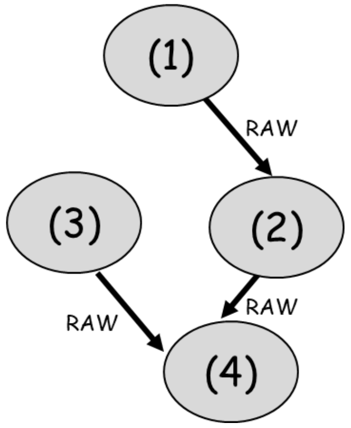
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

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

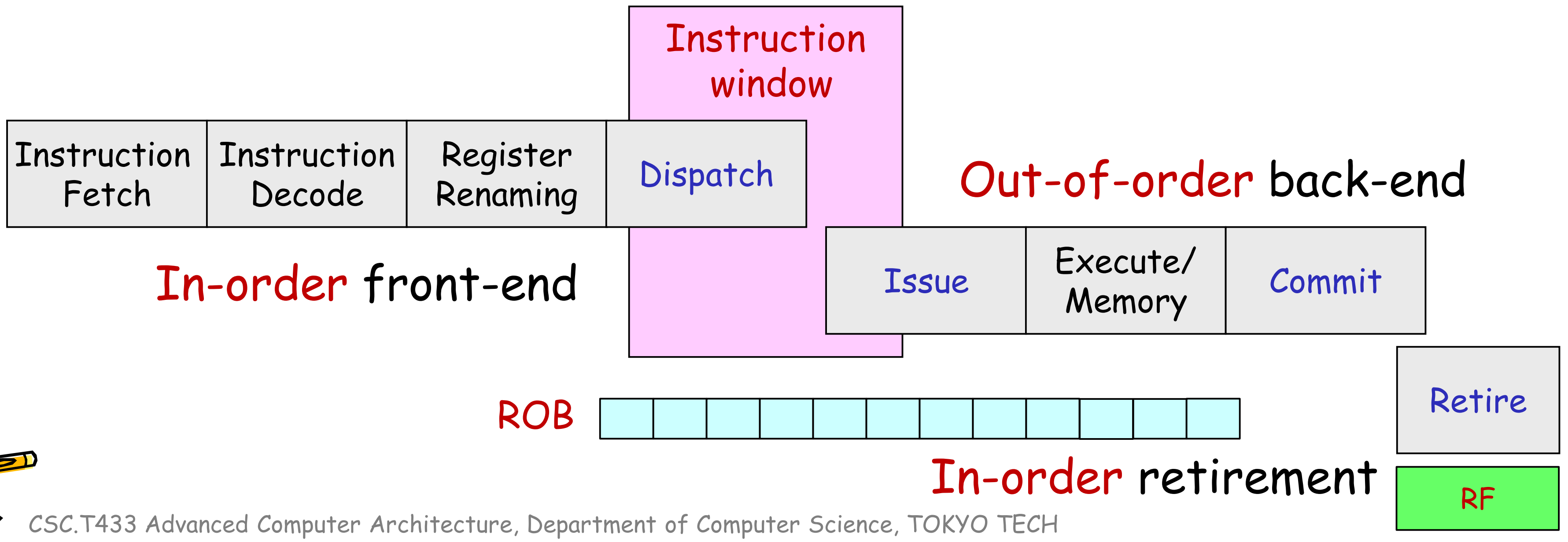
```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

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



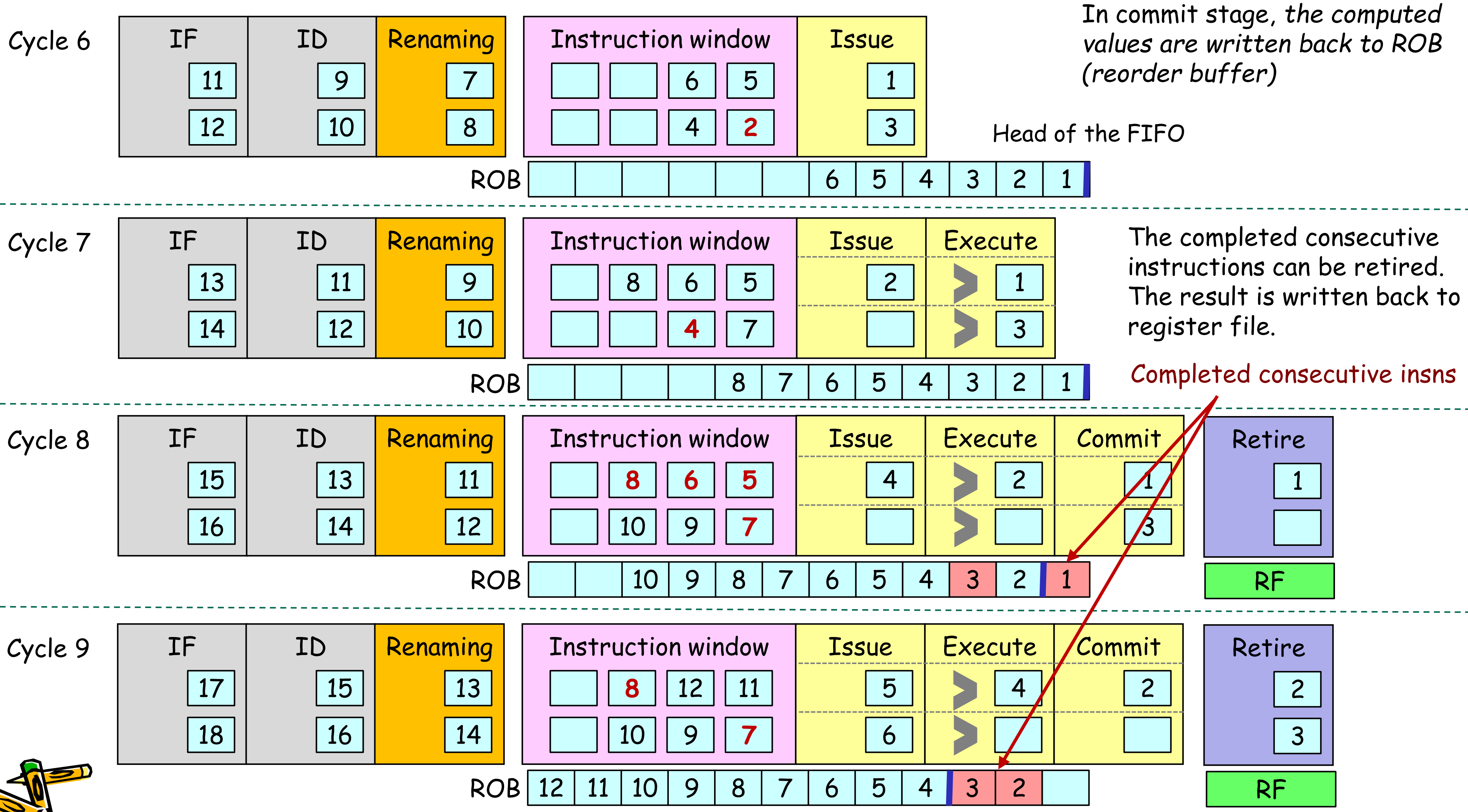
Instruction pipeline of OoO execution processor

- 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 of 32 registers*) using a logical register number from x0 to x31.



The key idea for OoO execution (last lecture)

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

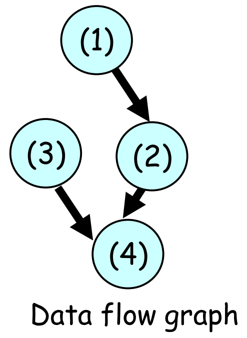


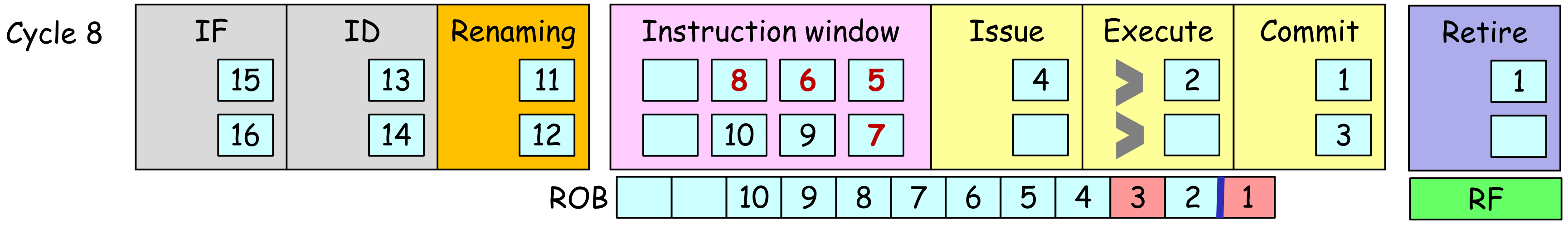
Cycle 6
IF 11 12
ID 9 10
Renaming 7 8
Instruction window 6 5 4 2
Issue 1 3
In commit stage, the computed values are written back to ROB (reorder buffer)
Head of the FIFO
ROB 6 5 4 3 2 1
Cycle 7
IF 13 14
ID 11 12
Renaming 9 10
Instruction window 8 6 5 4 7
Issue 2
Execute 1 3
The completed consecutive instructions can be retired. The result is written back to register file.
Completed consecutive insns
ROB 8 7 6 5 4 3 2 1
Cycle 8
IF 15 16
ID 13 14
Renaming 11 12
Instruction window 8 6 5 10 9 7
Issue 4
Execute 2
Commit 1 3
Retire 1
RF
ROB 10 9 8 7 6 5 4 3 2 1
Cycle 9
IF 17 18
ID 15 16
Renaming 13 14
Instruction window 8 12 11 10 9 7
Issue 5 6
Execute 4
Commit 2
Retire 2 3
RF
ROB 12 11 10 9 8 7 6 5 4 3 2

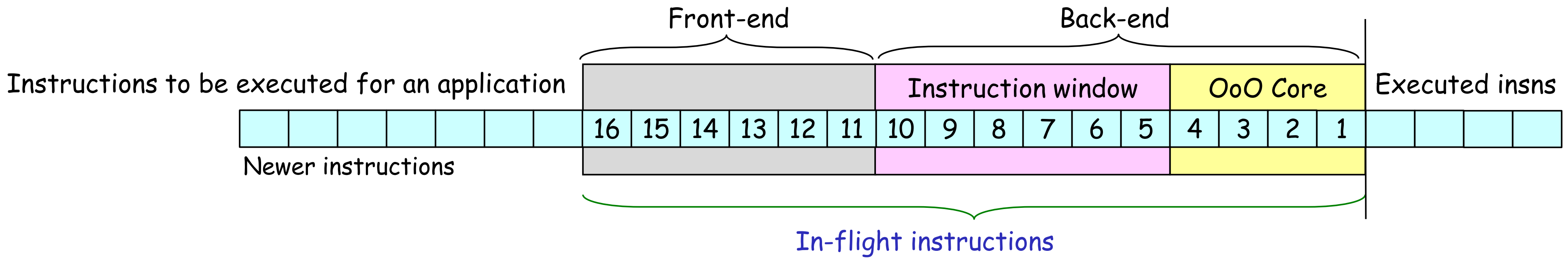
Architectural register file

Register dataflow

- In-flight instructions are ones processing in a processor



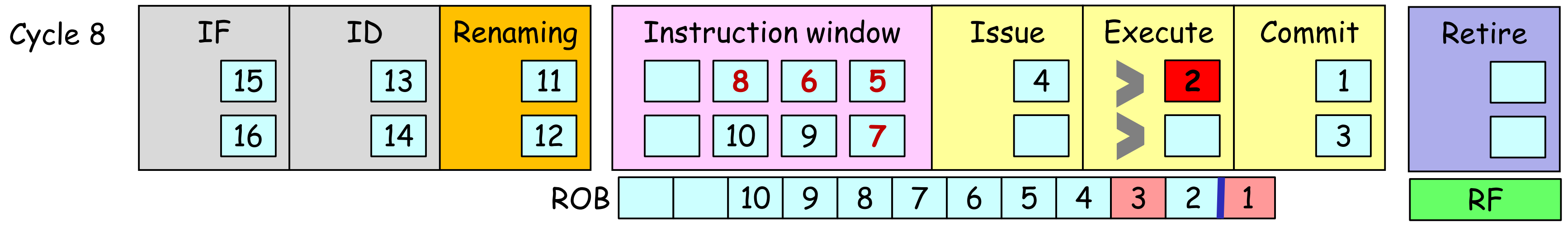


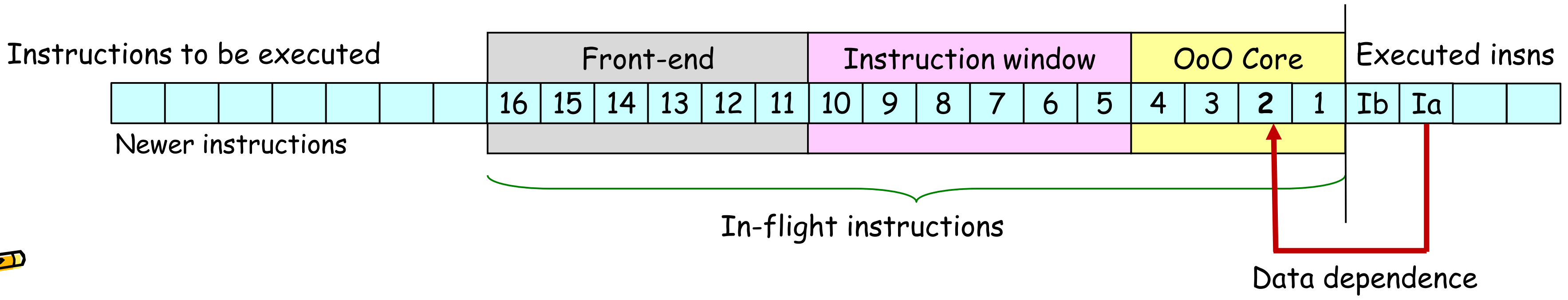


Case 1: Register dataflow from a far previous instn

- One source operand of insn I2 is from a retired instruction Ia.
- Because Ia is retired long ago, the physical destination register has been freed. The tag of the source register x3 can not be renamed at the renaming stage for I2, still having the logical register tag x3.
- Where does the operand x3 of I2 come from?

```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
```

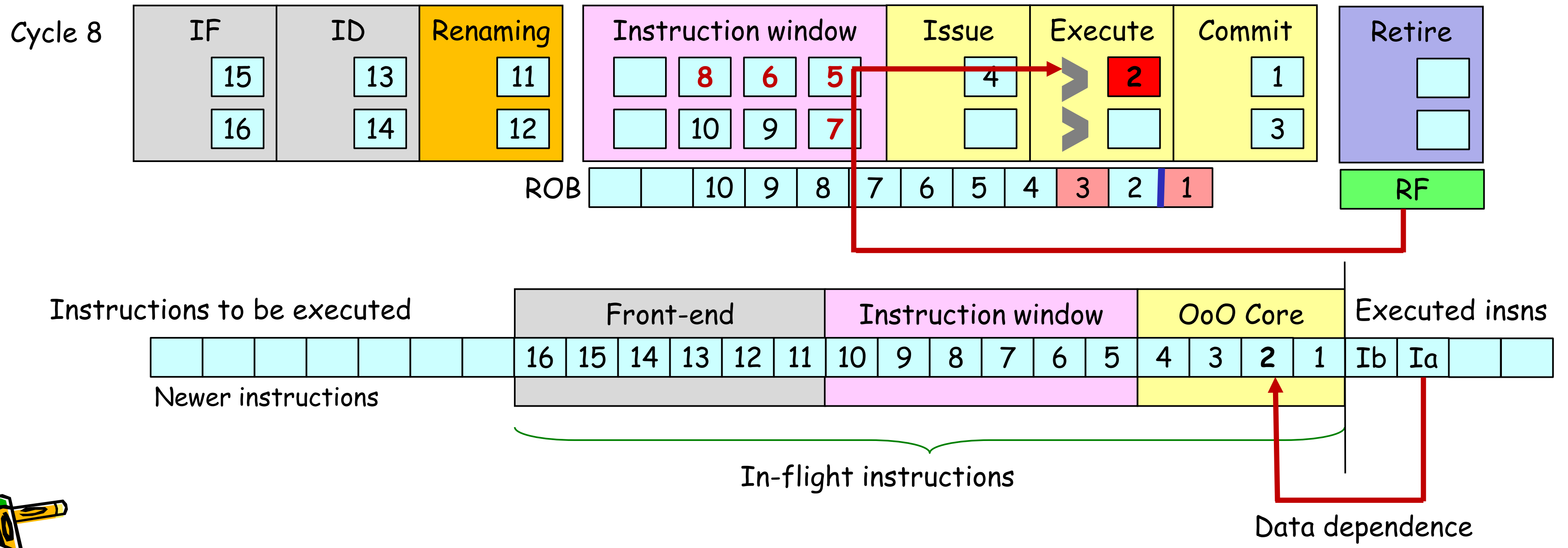




Case 1: Register dataflow from RF

- One source operand of insn I2 is from a retired instruction Ia.
- Because Ia is retired long ago, the physical destination register has been freed. The tag of the source register x3 can not be renamed at the renaming stage for I2, still having the logical register tag x3.
- Where does the operand x3 of I2 come from?

```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
```

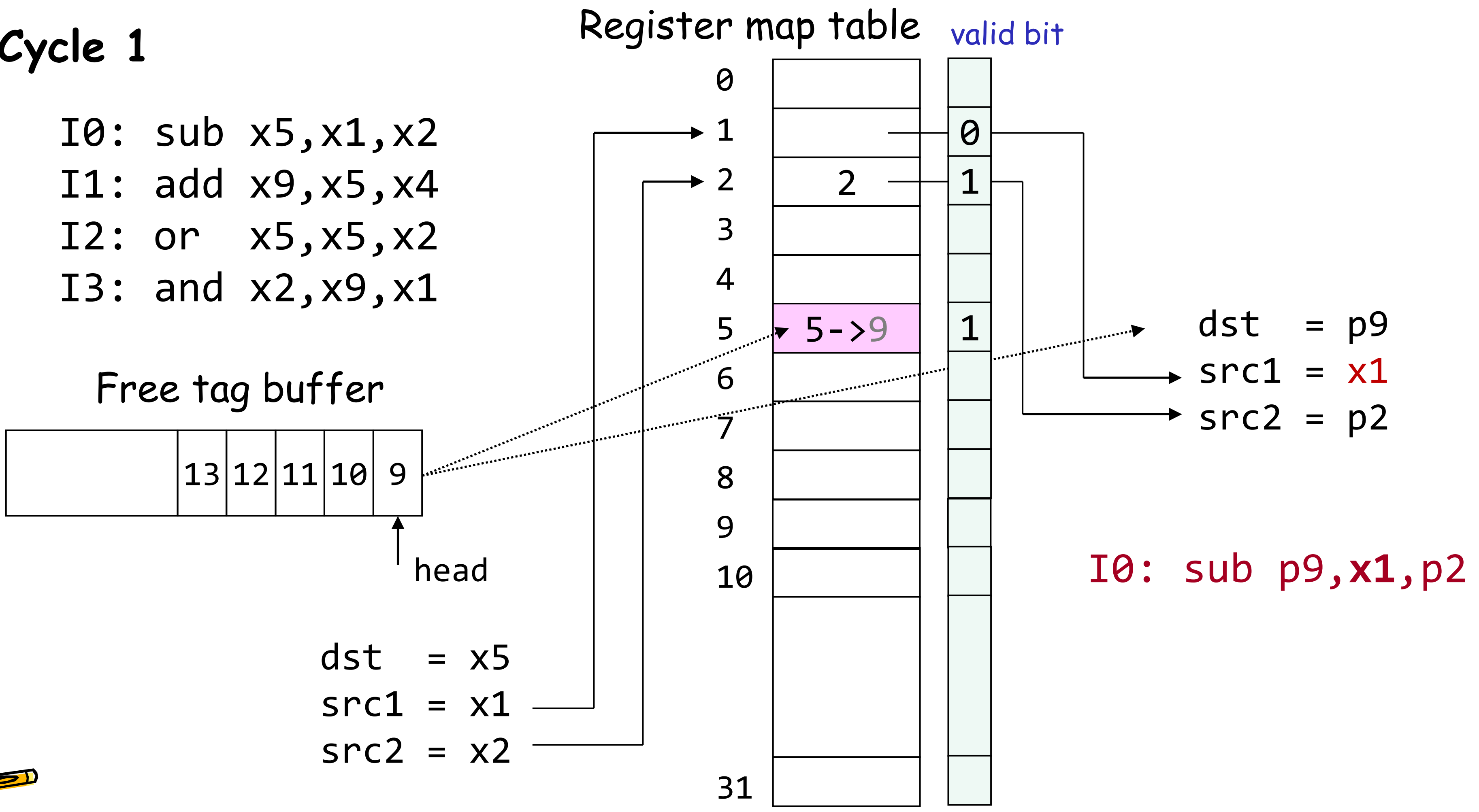


Example behavior of register renaming and valid bit

- A processor remembers a set of renamed logical registers.
- If x1 and x2 are not renamed for in-flight insn, it uses x1 and x2 instead of p1 and p2.

Cycle 1

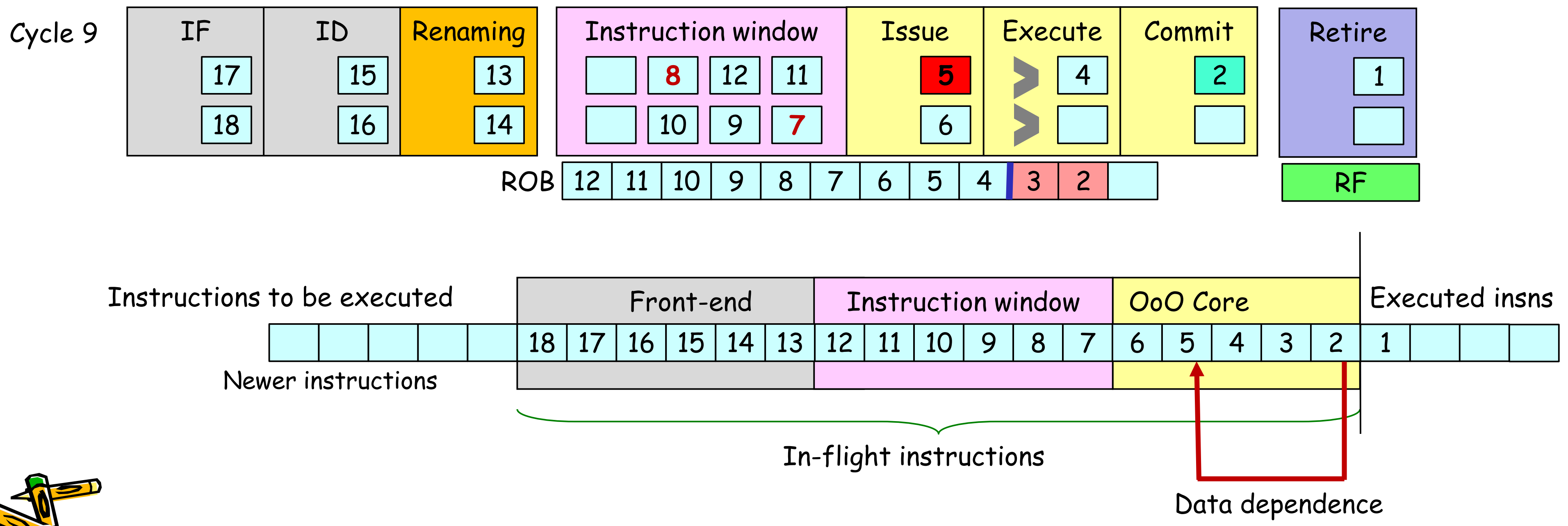
```
I0: sub x5,x1,x2
I1: add x9,x5,x4
I2: or  x5,x5,x2
I3: and x2,x9,x1
```



Case 2: Register dataflow

- Assume that one source operand p10 of insn I5 is from I2 which is not retired. The operand is generated a few clock cycles (tens of cycles sometimes) earlier.
- Because I2 is not retired, RF does not have the operand. Because I2 is committed, the operand is stored in ROB.
- Where does the operand of I5 come from?

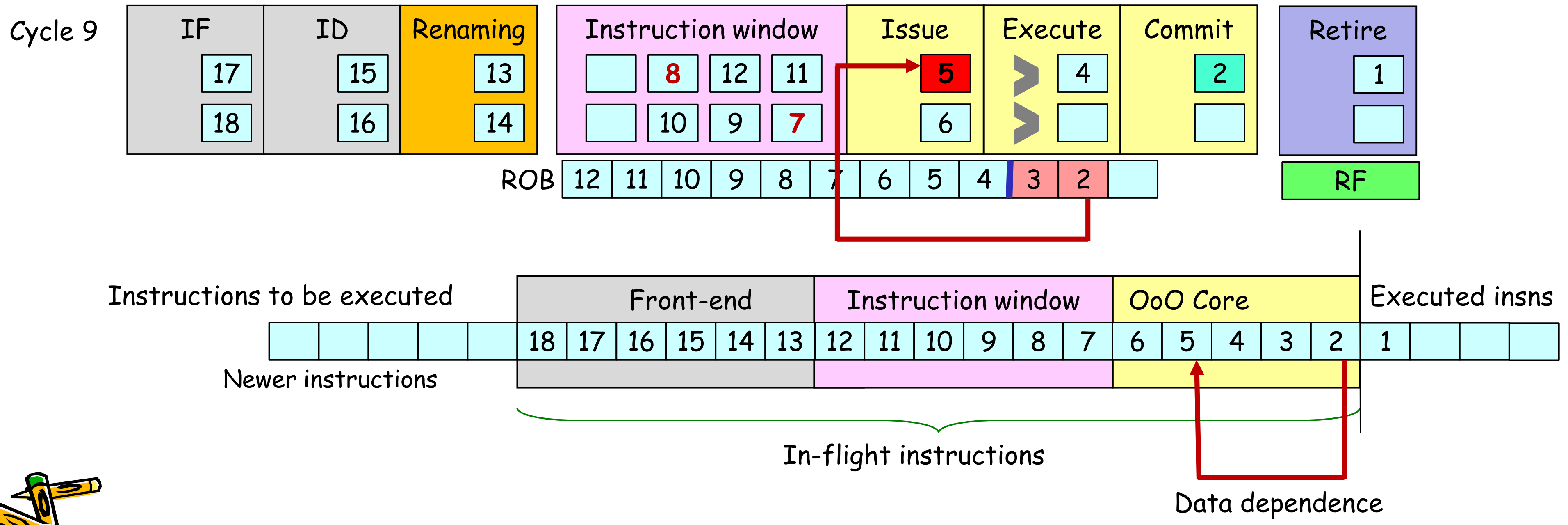
```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```



Case 2: Register dataflow from ROB

- Assume that one source operand p10 of insn I5 is from I2 which is not retired. The operand is generated a few clock cycles (tens of cycles sometimes) earlier.
- Because I2 is not retired, RF does not have the operand. Because I2 is committed, the operand is stored in ROB.
- Where does the operand of I5 come from?

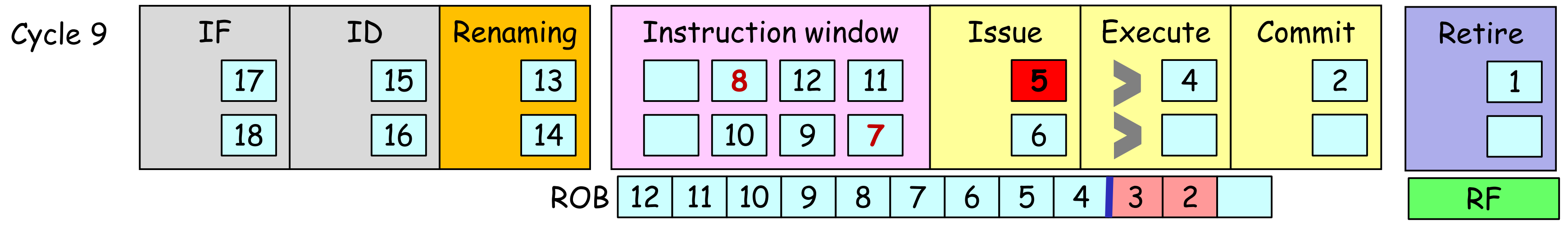
```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

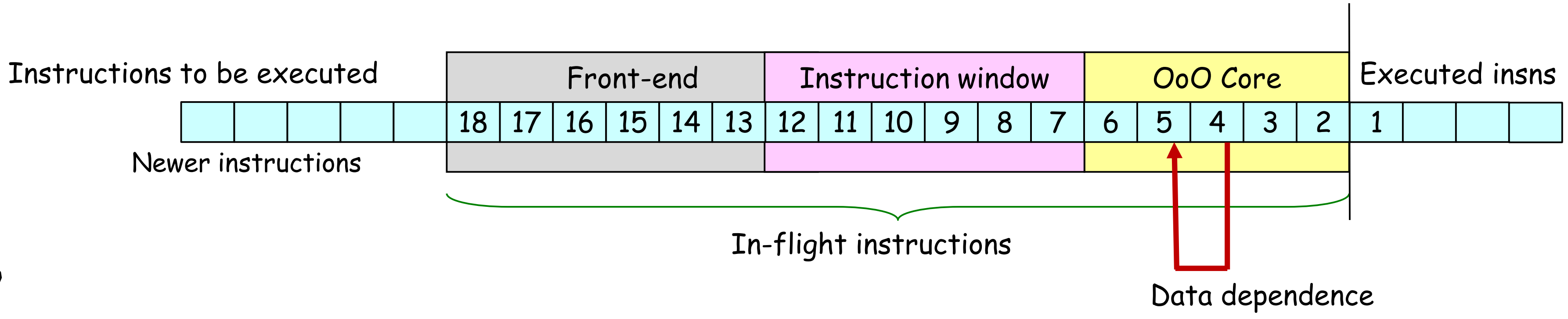


Case 3: Register dataflow

- Assume that the other source operand p12 of insn I5 is from I4 which is not committed. The operand is generated in the previous clock cycle.
- Because I4 is not retired, RF does not have the operand.
  Because I4 is not committed, ROB does not have the operand.
- Where does the operand of I5 come from?

```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

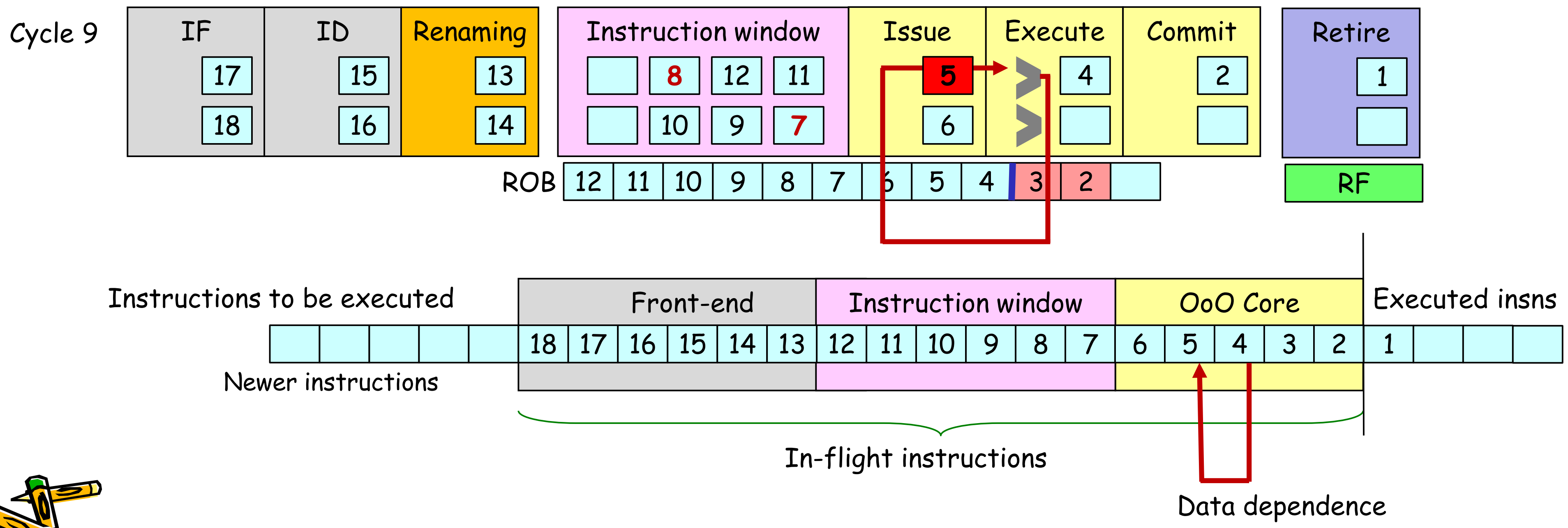




Case 3: Register dataflow from ALUs

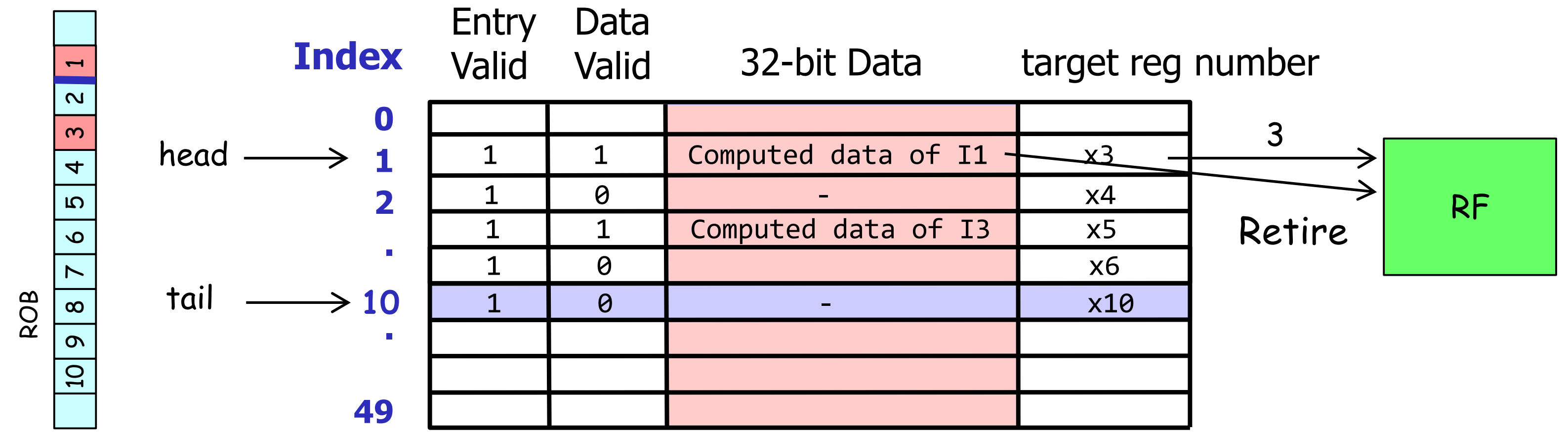
- Assume that the other source operand p12 of insn I5 is from I4 which is not committed. The operand is generated in the previous clock cycle.
- Because I4 is not retired, RF does not have the operand.
  Because I4 is not committed, ROB does not have the operand.
- Where does the operand of I5 come from?

```
Ia: add x3,x0,x0
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```



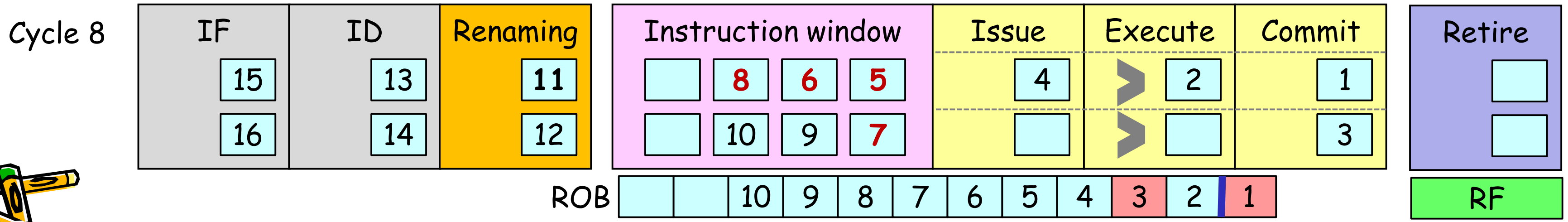
Reorder buffer (ROB)

- Each ROB entry has following fields
  - entry valid bit, data valid bit, **data**, target register number, etc.
- ROB provides the large physical registers for renaming
  - in fact, physical register number is ROB entry number
- The value of a physical register is from a matching ROB entry

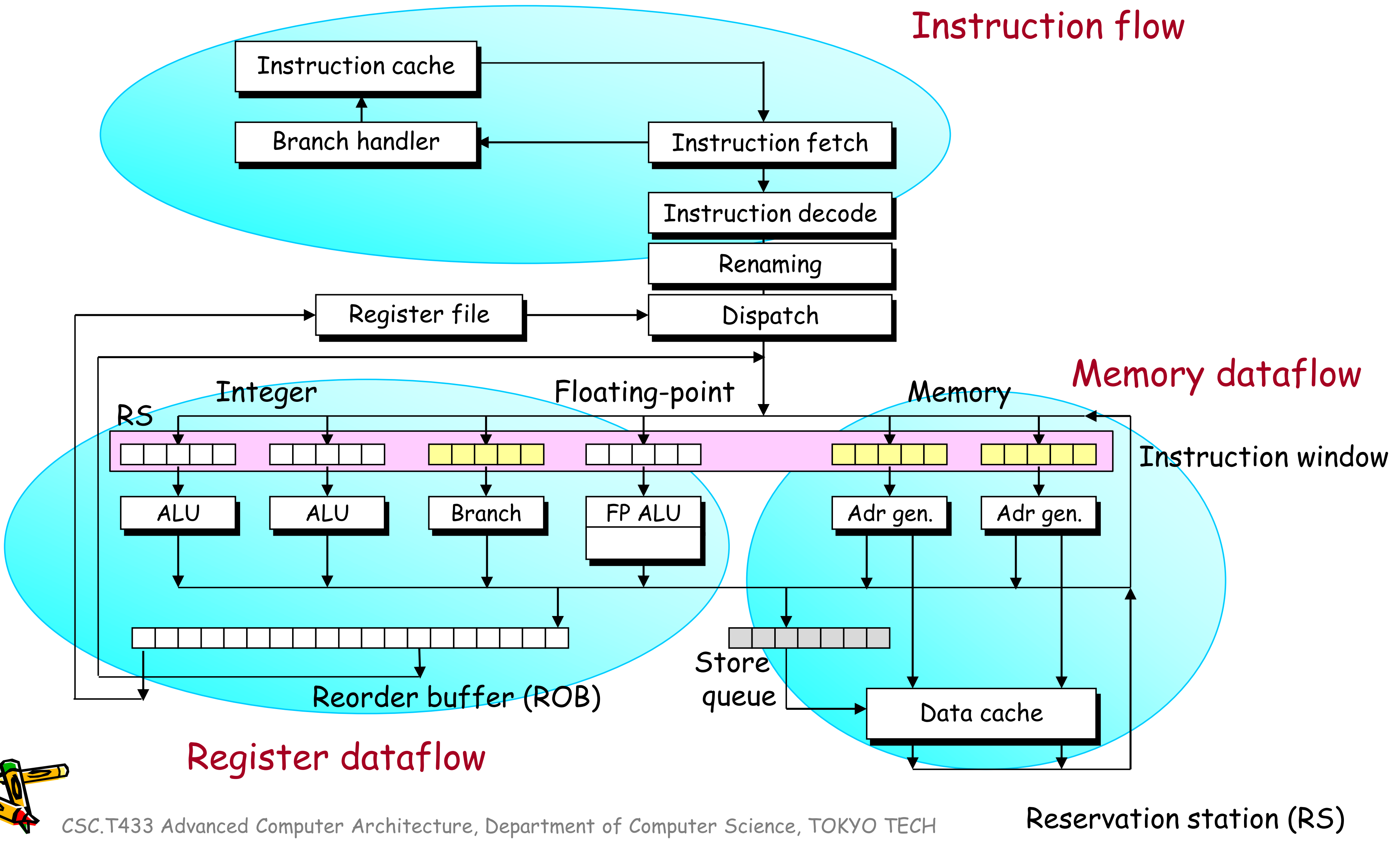


Index	Entry Valid	Data Valid	32-bit Data	target reg number
0				
1	1	1	Computed data of I1	x3
2	1	0	-	x4
.	1	1	Computed data of I3	x5
	1	0		x6
10	1	0	-	x10
.				
49				

`I10: add p10,`**`p3`**`,p8 (add x10,x5,x6)`

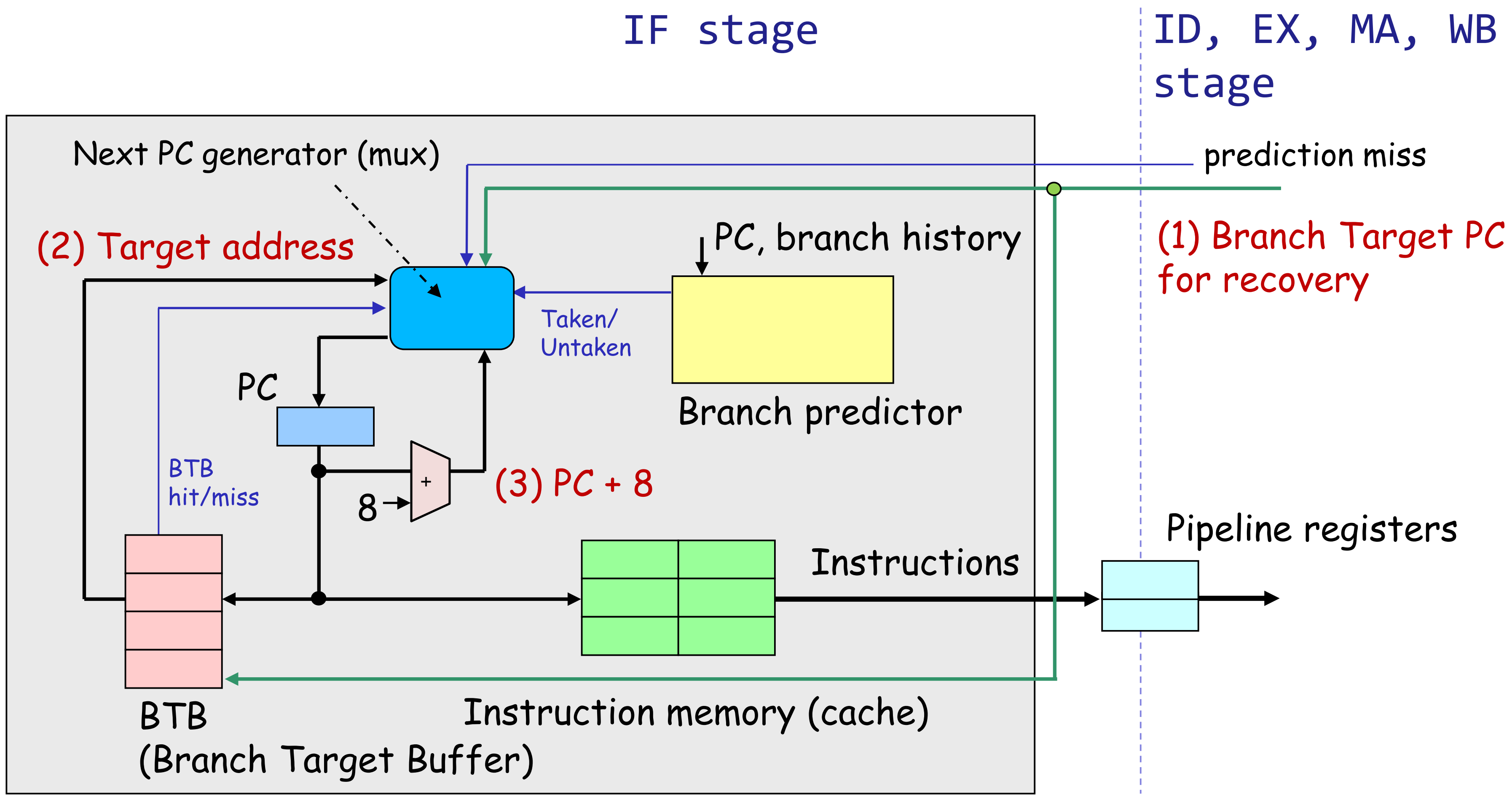


Datapath of OoO execution processor



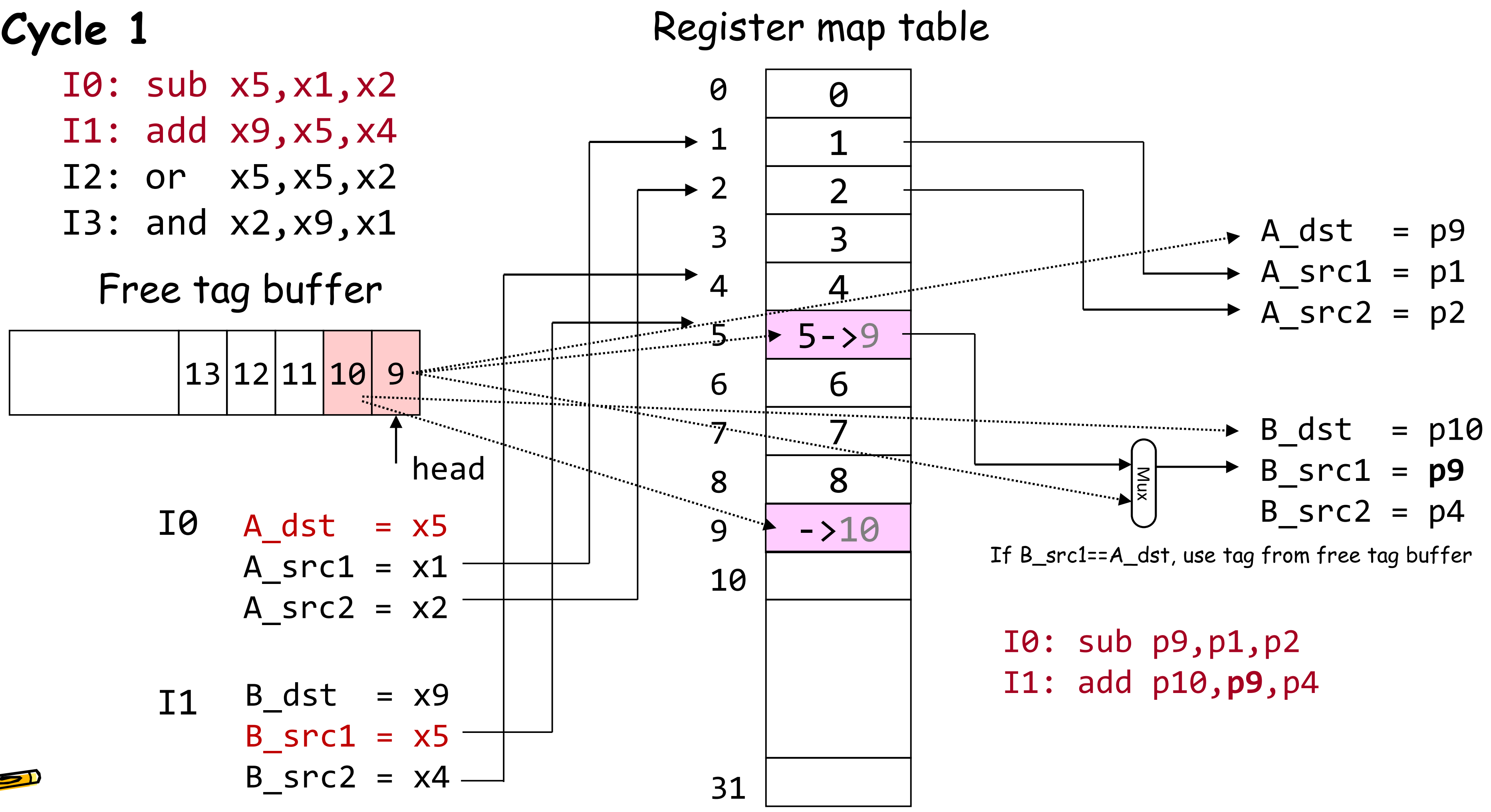
Instruction fetch unit of 2-way super-scalar

- High-bandwidth instruction delivery using prediction, and speculation



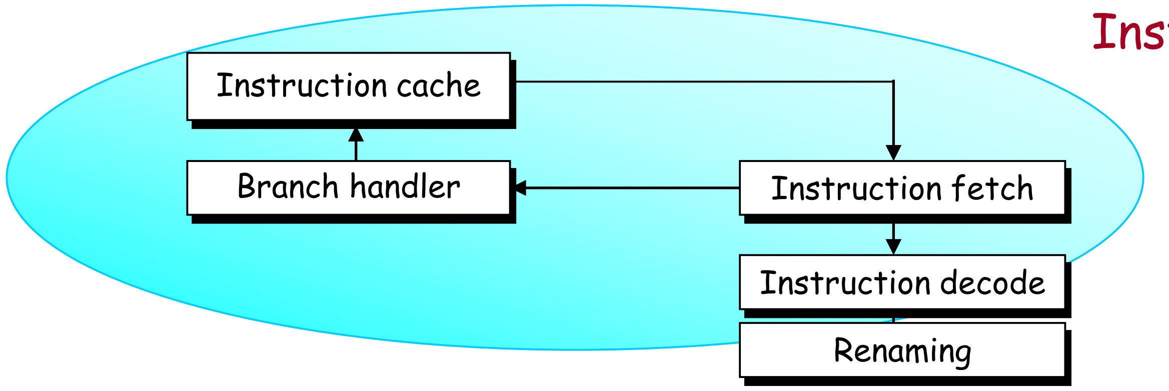
Renaming two instructions per cycle for superscalar

- Renaming instruction I0 and I1



Datapath of OoO execution processor (partially)

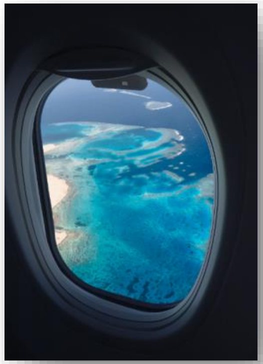
Instruction flow

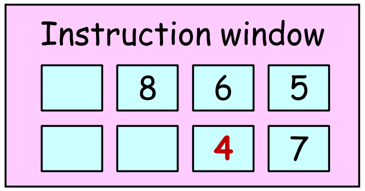


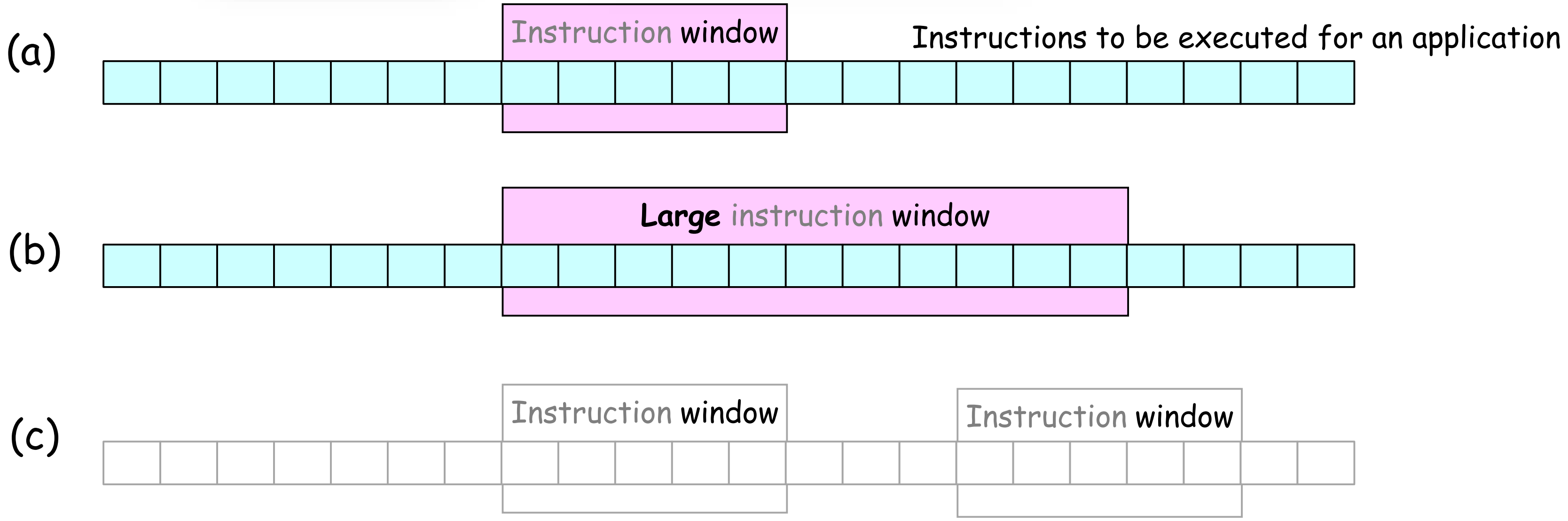
Reservation station (RS)

Aside: What is a window?

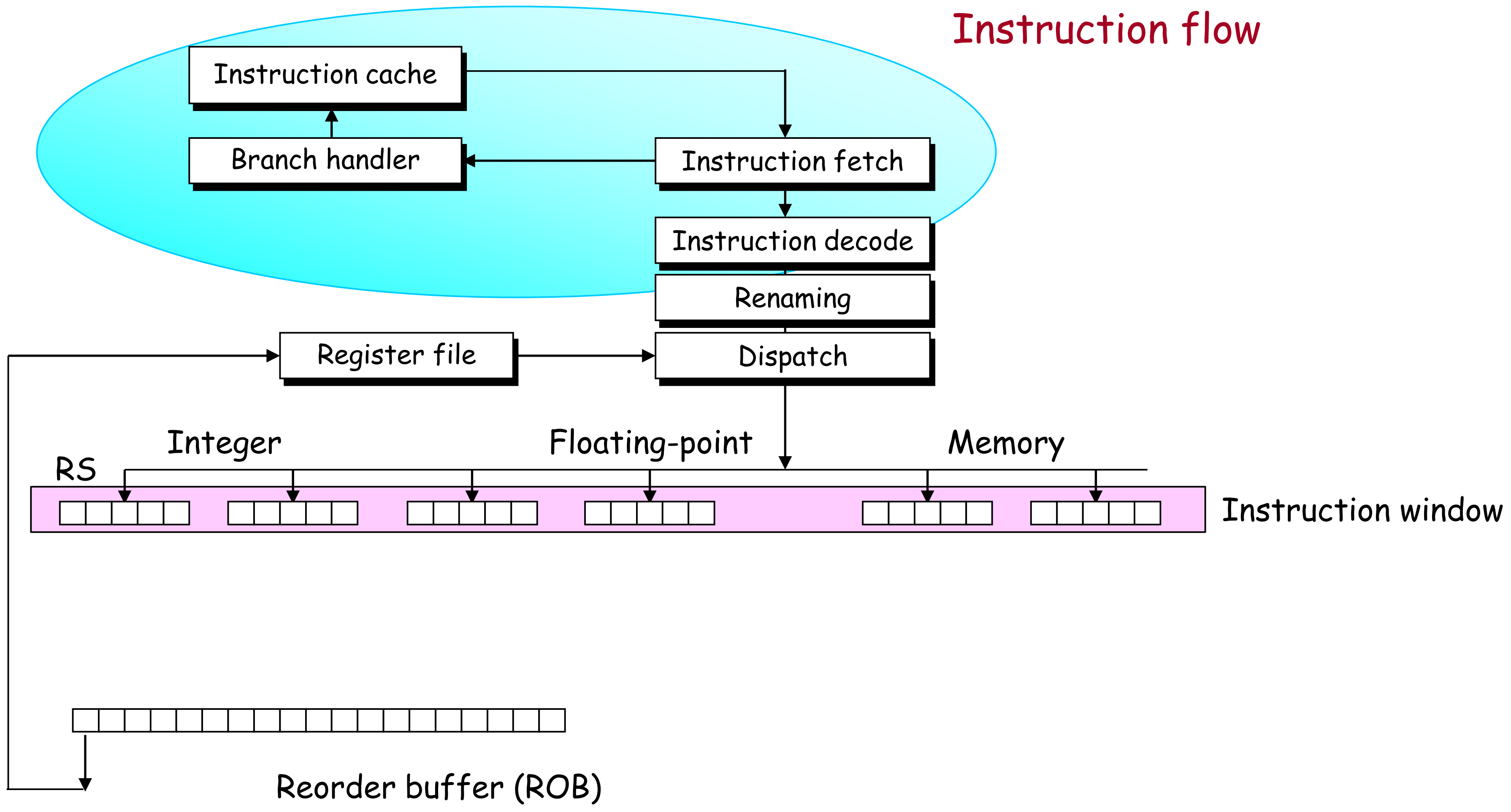
- A window is a space in the wall of a building or in the side of a vehicle, which has glass in it so that light can come in and you can see out. (Collins)







Datapath of OoO execution processor (partially)

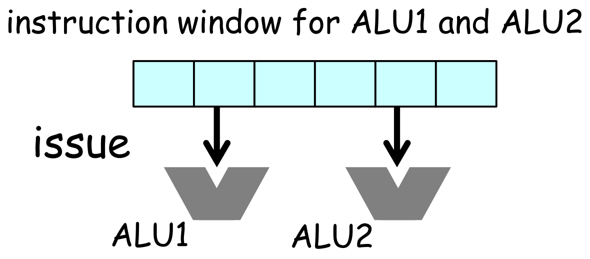


Register dataflow

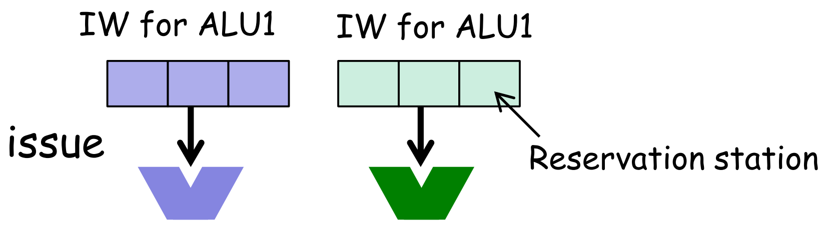
Reservation station (RS)

Reservation station (RS)

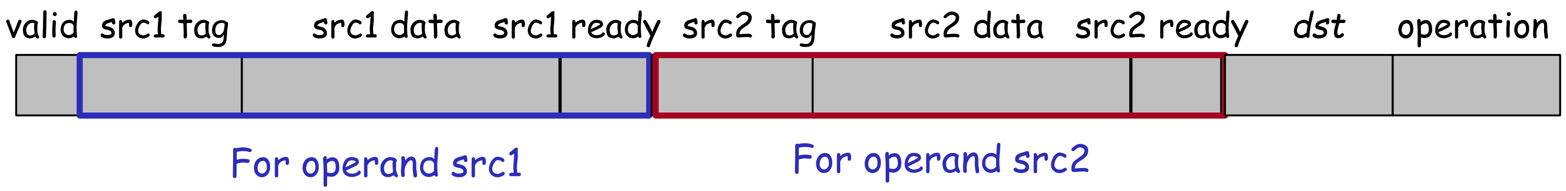
- To simplify the wakeup and select logic at issue stage, each functional unit (ALU) has own instruction window, an entry for an instruction is called reservation station (RS).
- Each reservation station has
  - entry valid bit, src1 tag, src1 data, src1 ready, src2 tag, src2 data, src2 ready, destination physical register number (*dst*), operation, …
  - The computed data (outcome) with its *dst* as tag is broadcasted to all RSs.



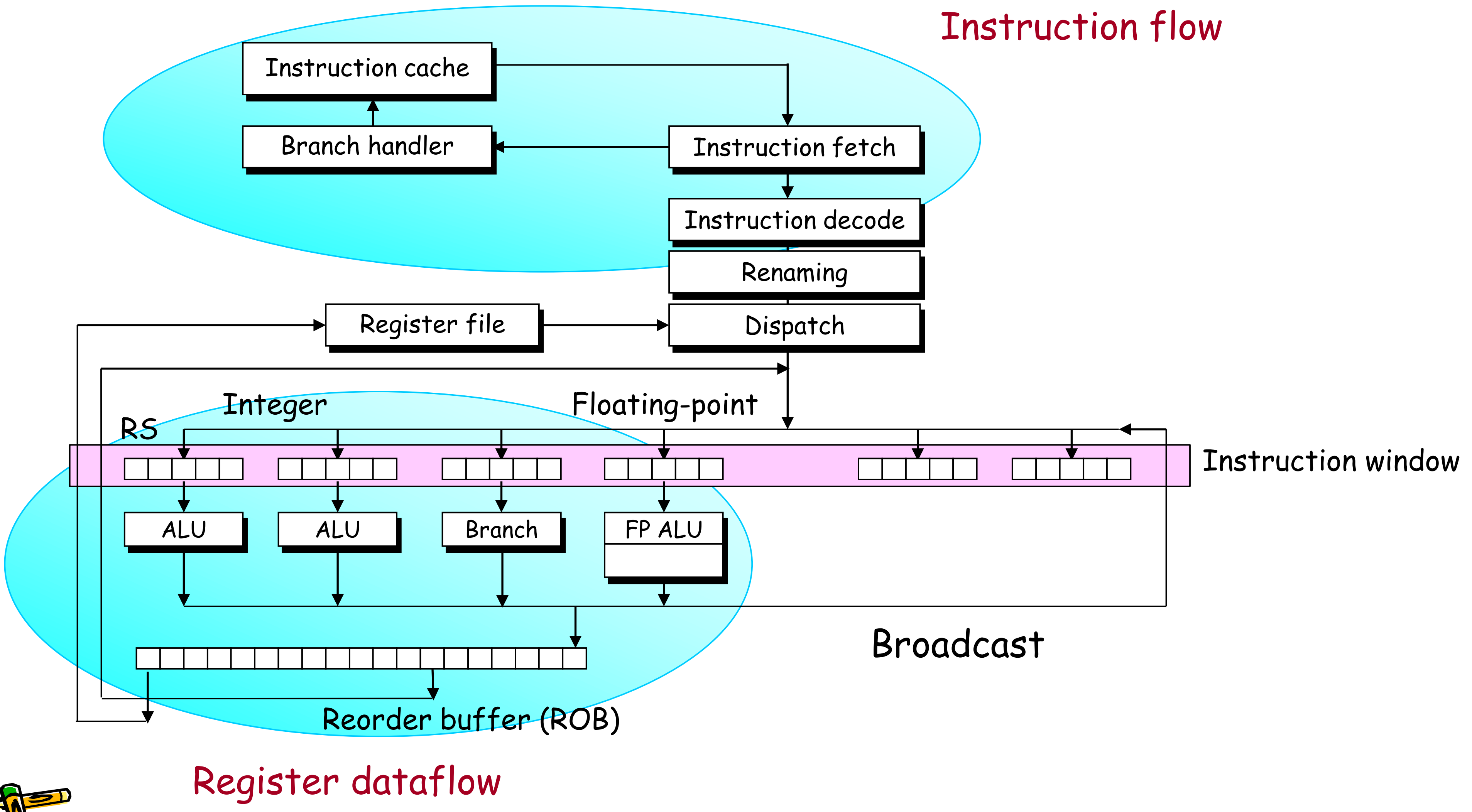
(a) Central instruction window



(b) instruction window using RS



Datapath of OoO execution processor (partially)

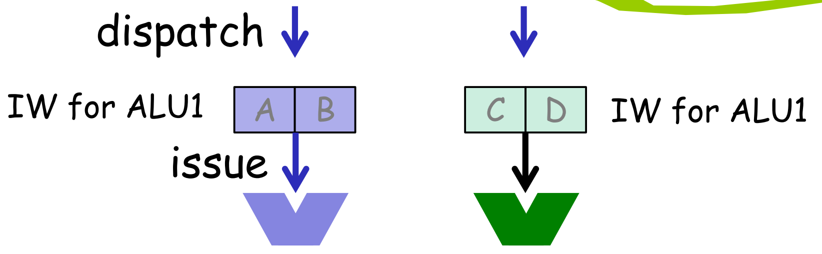


Reservation station (RS)

Example behavior of reservation stations

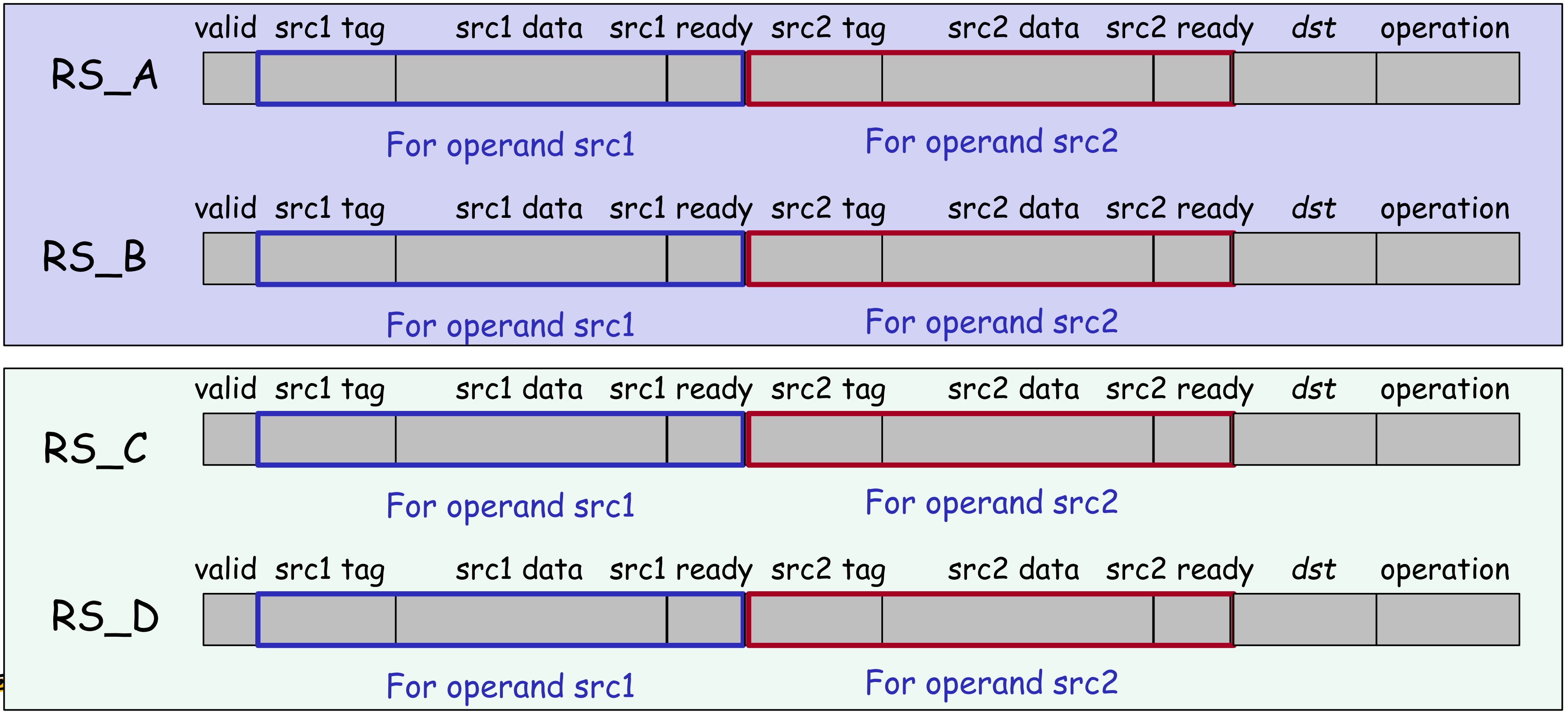
Cycle 0

dispatch I1, I2



```
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

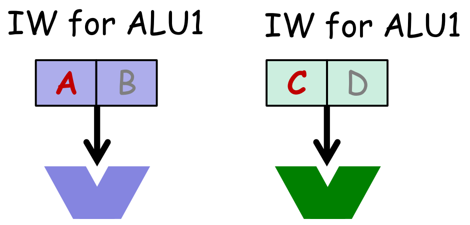
dispatch at most two instructions, one to A or B and the other to C or D



Example behavior of reservation stations

Cycle 1

dispatch I3, I4
issue I1



```
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

dispatch at most two instructions, one to A or B and the other to C or D

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A	1	x1	value of x1	1	x2	value of x2	1	p9	I1: sub
		For operand src1			For operand src2				

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_B									
		For operand src1			For operand src2				

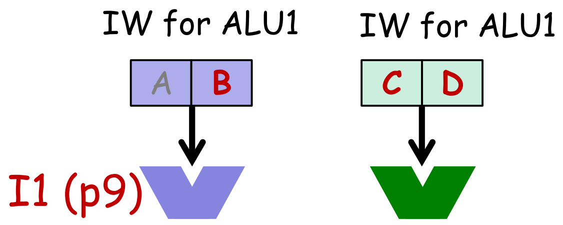
	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C	1	p9		0	x3	value of x3	1	p10	I2: add
		For operand src1			For operand src2				

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_D									
		For operand src1			For operand src2				

Example behavior of reservation stations

Cycle 2

dispatch I5, I6
issue I2, I3
execute I1



```
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

dispatch at most two instructions, one to A or B and the other to C or D

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A	1/0								I1: sub
		For operand src1			For operand src2				

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_B	1	x4	value of x4	1	x5	value of x5	1	p11	I3: or
		For operand src1			For operand src2				

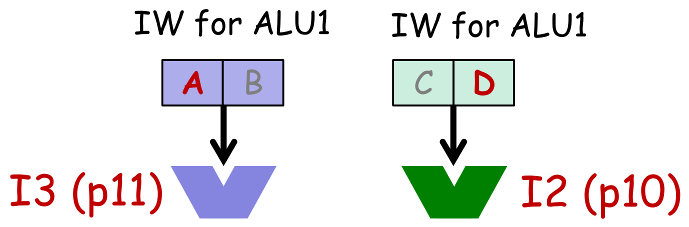
	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C	1	p9	value of p9	1	x3	value of x3	1	p10	I2: add
		For operand src1			For operand src2				

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_D	1	p10		0	p11		0	p12	I4: and
		For operand src1			For operand src2				

Example behavior of reservation stations

Cycle 3

dispatch I7, I8
issue I4
execute I2, I3



```
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,x4,x5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

dispatch at most two instructions, one to A or B and the other to C or D

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	dst	operation
RS_A	1	p10	value of p10	1	p12		0	p13	I5: nor

For operand src1 For operand src2

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	dst	operation
RS_B	1/0								I3: or

For operand src1 For operand src2

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	dst	operation
RS_C	1/0								I2: add

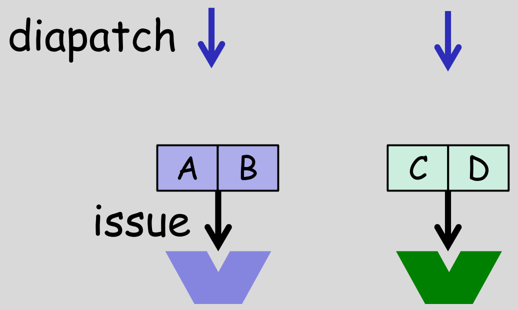
For operand src1 For operand src2

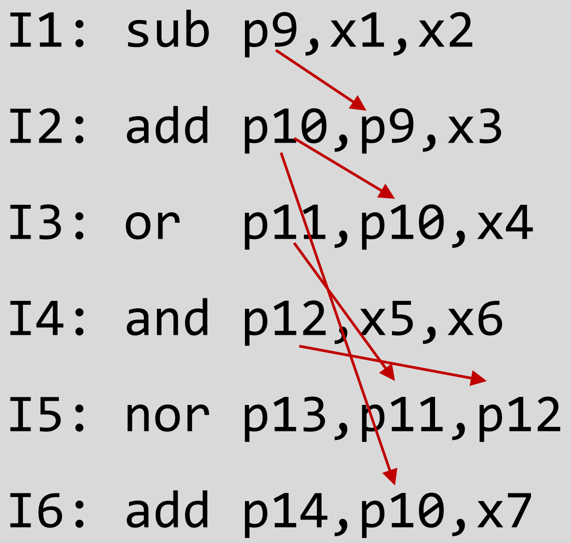
	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	dst	operation
RS_D	1	p10	value of p10	1	p11	value of p11	1	p12	I4: and

For operand src1 For operand src2

Exercise 1

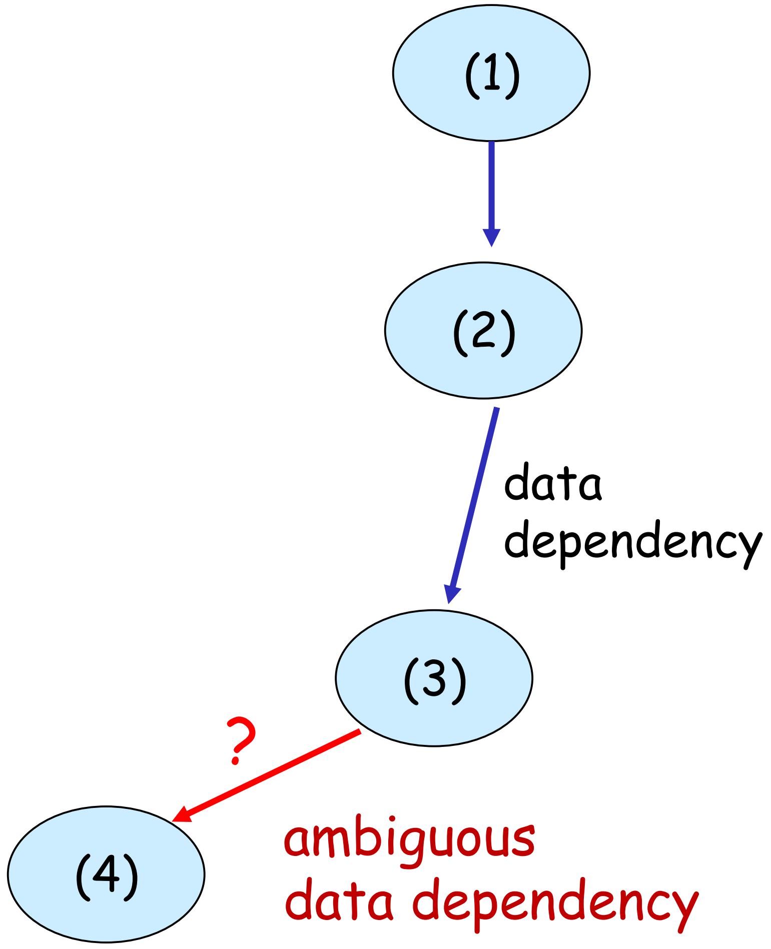
- Example behavior of reservation stations




```
I1: sub p9,x1,x2
I2: add p10,p9,x3
I3: or  p11,p10,x4
I4: and p12,x5,x6
I5: nor p13,p11,p12
I6: add p14,p10,x7
```

Instruction Level Parallelism (ILP)

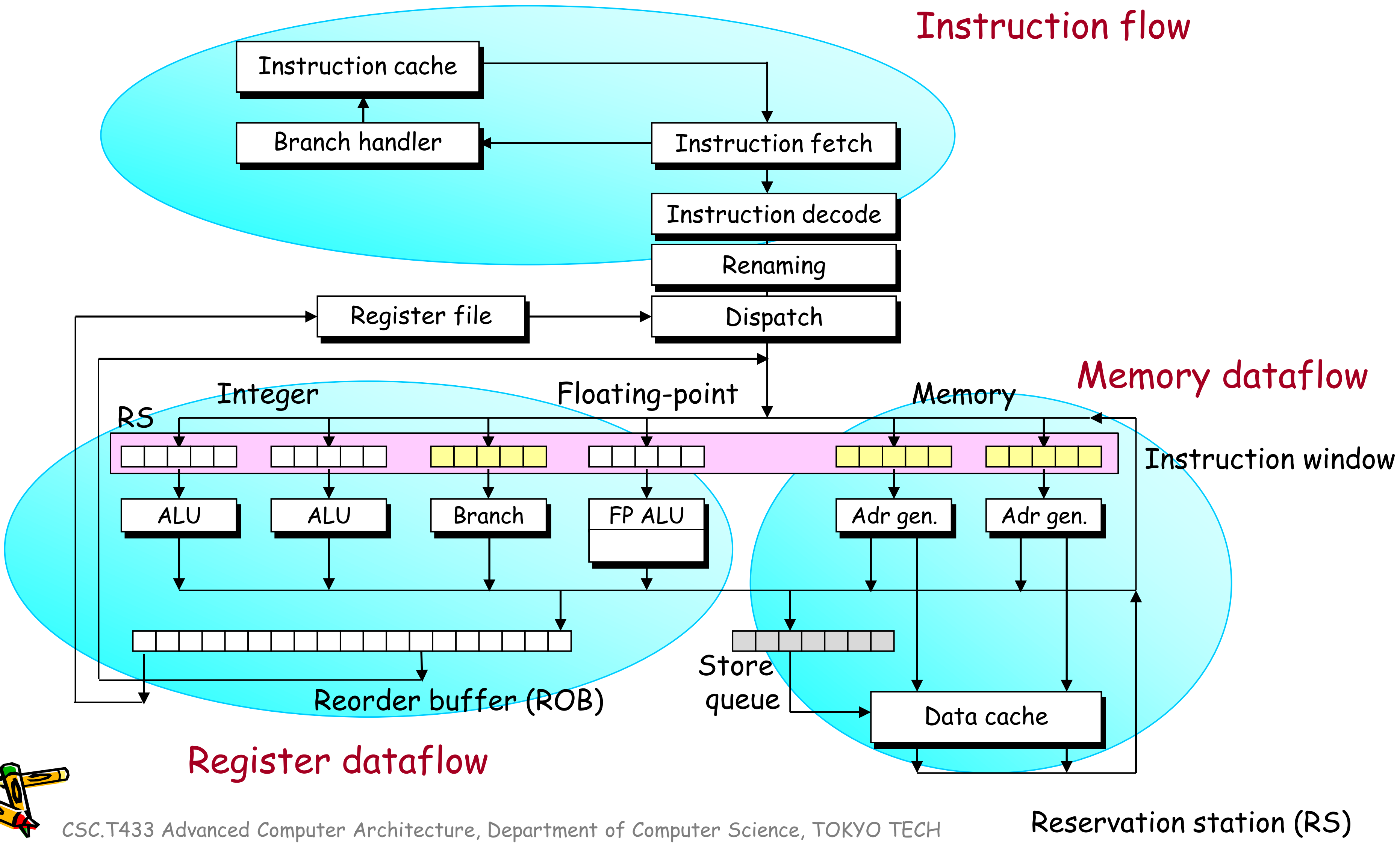
```
lw    t0, 32(s3)      (1)
add   t0, s2, t0      (2)
sw    t0, 48(s3)      (3)
                 ?
lw    t1, 32(s4)      (4)
```



Memory dataflow and branches

- The update of a data cache cannot be recovered easily. So, cache update is done at the retire stage in-order manner by using store queue.
  Because of the ambiguous memory dependency, load and store instructions can be executed in-order manner.
  - About 30% (or less) of executed instructions are load and stores.
  - Even if they are executed in-order, IPC of 3 can be achieved.
- Branch instructions can be executed in-order manner.
  - About 20% (or less) of executed instructions are jump and branch instructions.
  - Out-or-order branch execution and aggressive miss recovery may cause false recovery (recovery by a branch on the false control path).

Datapath of OoO execution processor



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

From multi-core era to many-core era

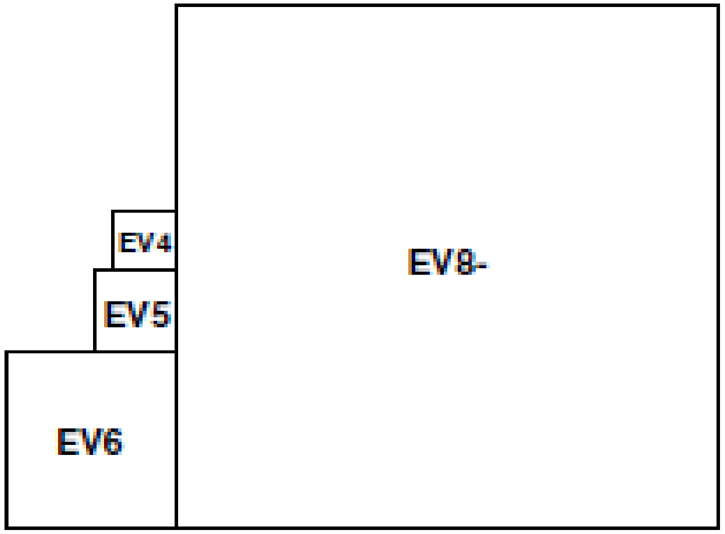
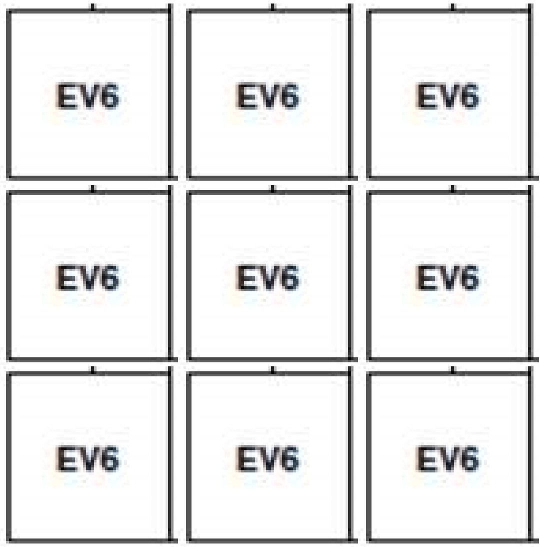


Figure 1. Relative sizes of the cores used in the study



Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, MICRO-36

From multi-core era to many-core era

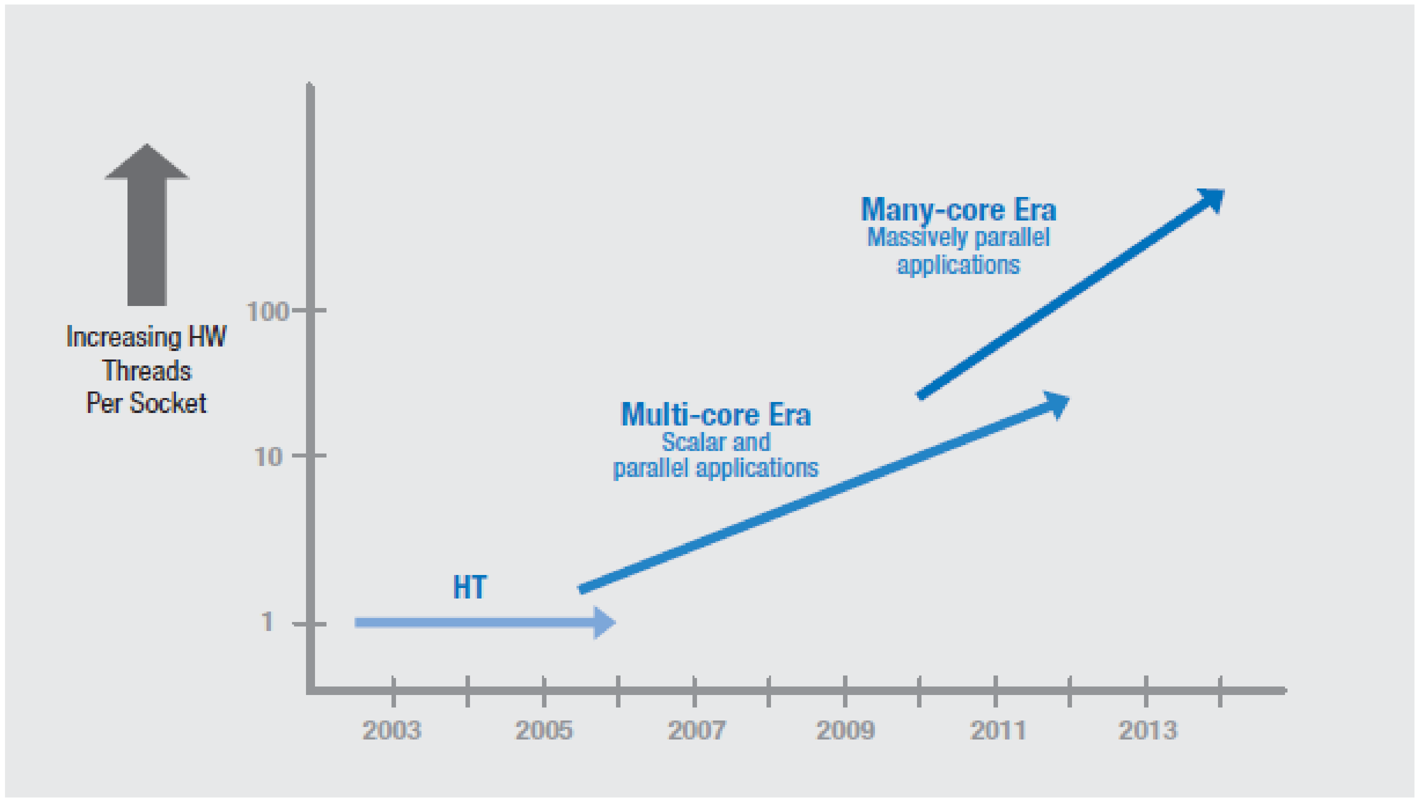
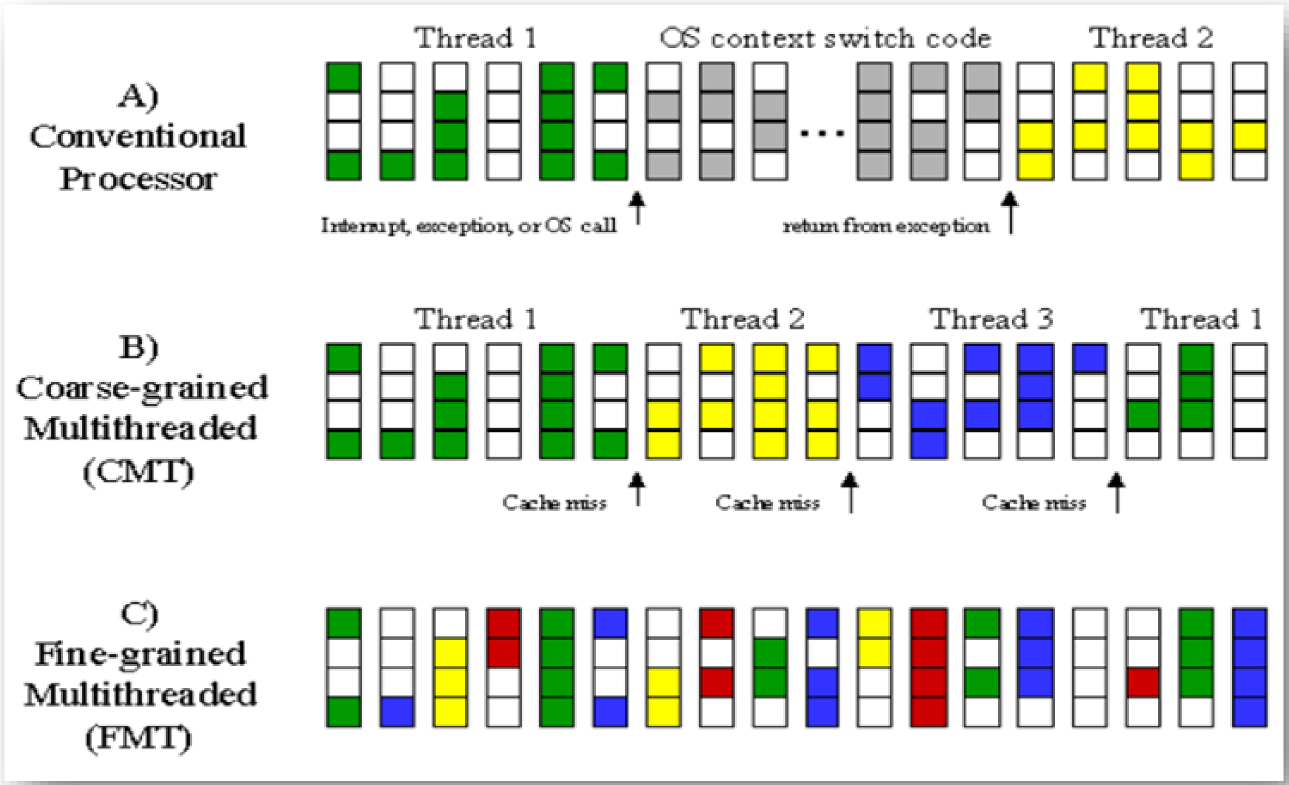


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Multithreading (1/2)

- During a branch miss recovery and access to the main memory by a cache miss, ALUs have no jobs to do and have to be idle.
- Executing multiple independent threads (programs) will mitigate the overhead.
- They are called coarse- and fine-grained multithreaded processors having multiple architecture states.


Thread 1
OS context switch code
Thread 2
A) Conventional Processor
Interrupt, exception, or OS call
return from exception
Thread 1
Thread 2
Thread 3
Thread 1
B) Coarse-grained Multithreaded (CMT)
Cache miss
Cache miss
Cache miss
C) Fine-grained Multithreaded (FMT)

http://www.realworldtech.com/alpha-ev8-smt/

Multithreading (2/2)

- Simultaneous Multithreading (SMT) can improve hardware resource usage.

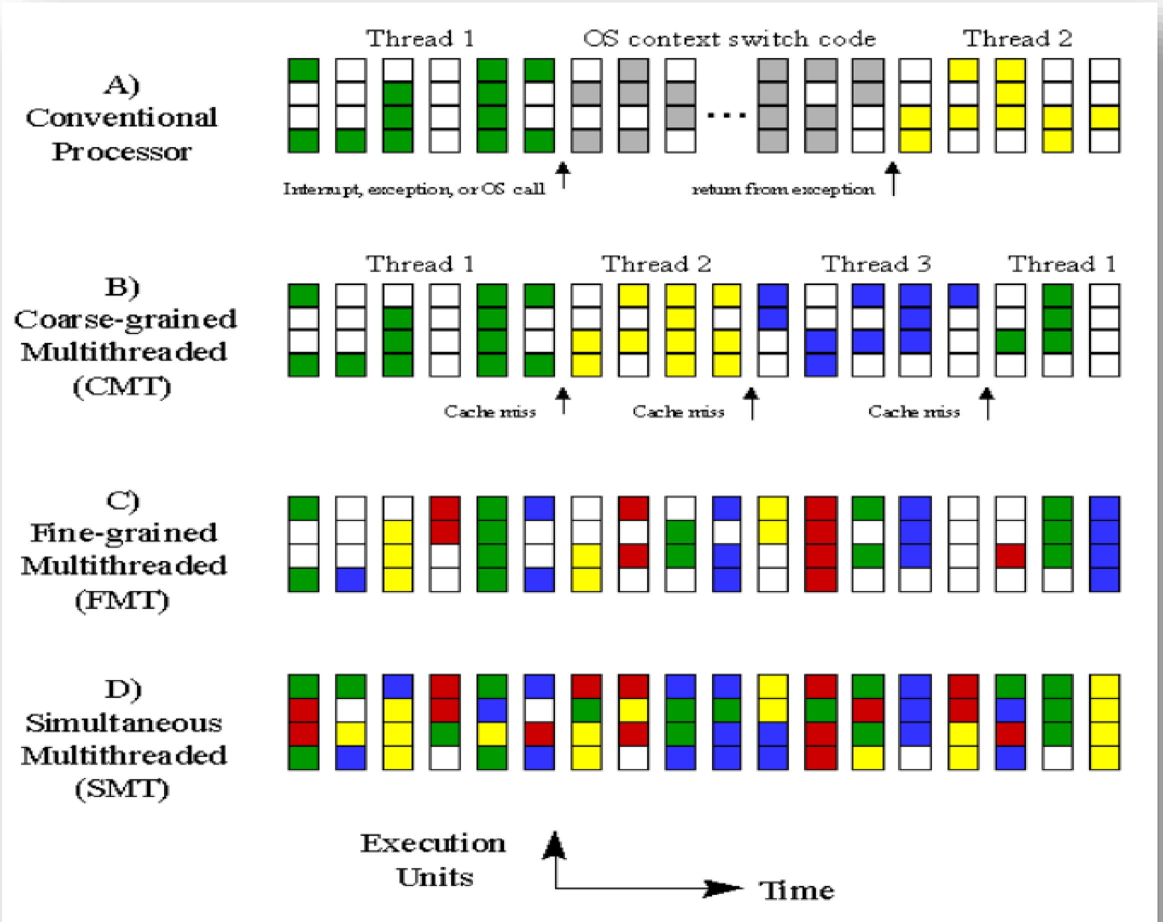

A) Conventional Processor
Thread 1
OS context switch code
Thread 2
Interrupt, exception, or OS call
return from exception
B) Coarse-grained Multithreaded (CMT)
Thread 1
Thread 2
Thread 3
Thread 1
Cache miss
Cache miss
Cache miss
C) Fine-grained Multithreaded (FMT)
D) Simultaneous Multithreaded (SMT)
Execution Units
Time

Figure 1. Multithreaded Execution with Increasing Levels of TLP Hardware Support

http://www.realworldtech.com/alpha-ev8-smt/

Exercise 1

Cycle 0 dispatch I1, I2

Cycle 1

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									

Cycle 2

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									

Cycle 3

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									

Exercise 1

Cycle 0 dispatch I1, I2

Cycle 4

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									

Cycle 5

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									

Cycle 6

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_A									
RS_B									

	valid	src1 tag	src1 data	src1 ready	src2 tag	src2 data	src2 ready	*dst*	operation
RS_C									
RS_D									