Fiscal Year 2019

Ver. 2020-01-05a

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

Advanced Computer Architecture

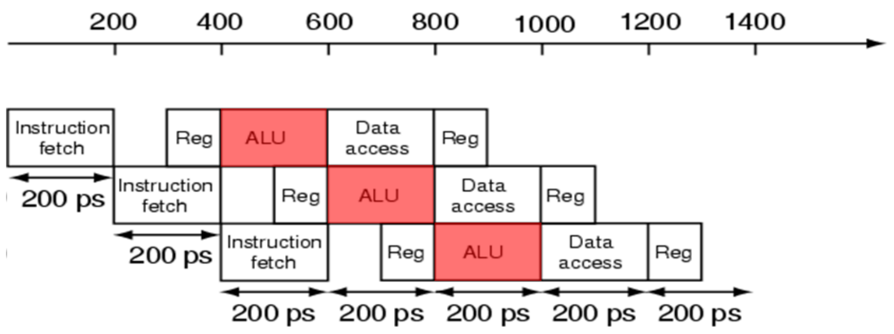
8. Instruction Level Parallelism: Dynamic Scheduling

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

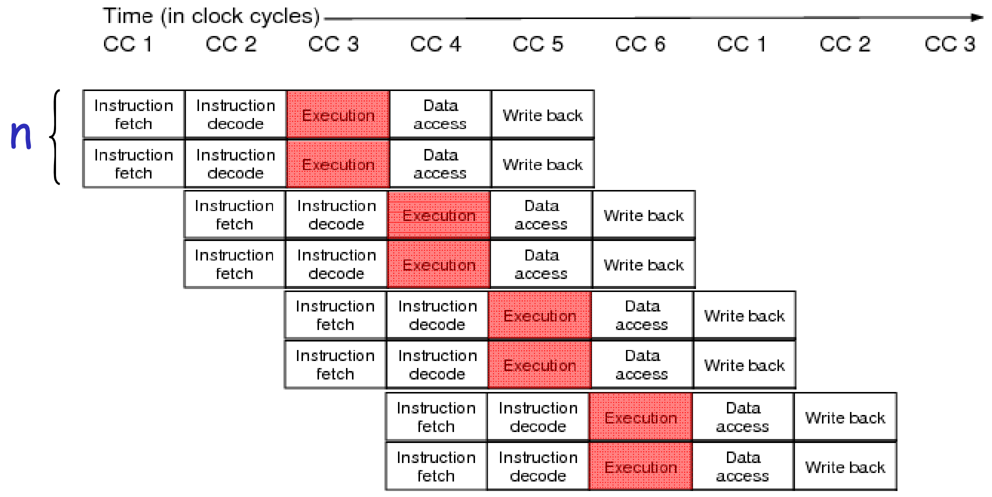
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Scalar and Superscalar processors

- Scalar processor can execute at most one single instruction per clock cycle using one ALU.
  - IPC (Executed Instructions Per Cycle) is less than 1.
- Superscalar processor can execute more than one instruction per clock cycle by executing multiple instructions using multiple pipelines.
  - IPC (Executed Instructions Per Cycle) can be more than 1.
  - using n pipelines is called n-way superscalar



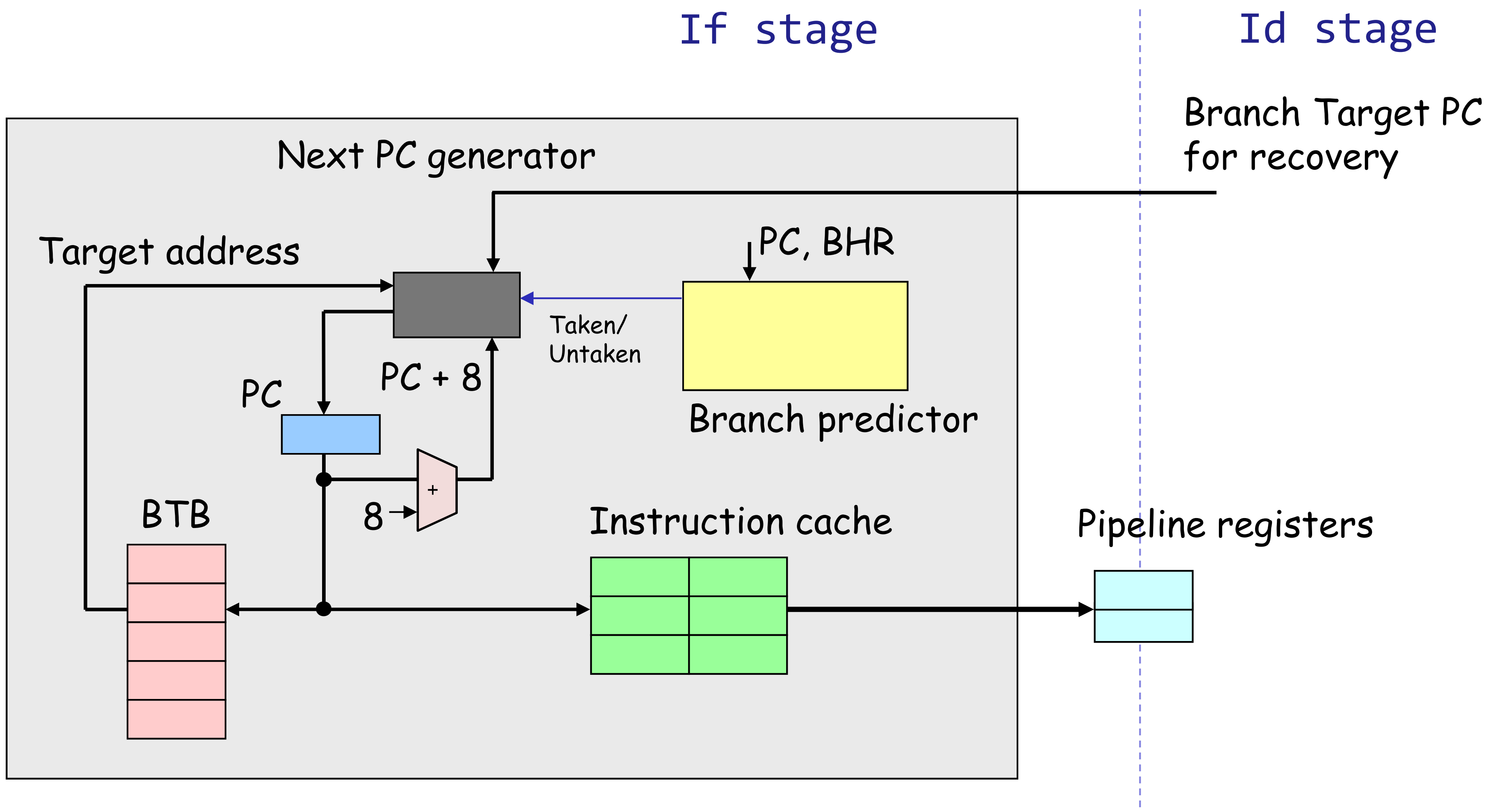
(a) pipeline diagram of scalar processor



(b) pipeline diagram of 2-way superscalar processor

Instruction fetch unit in IF stage

- For high-bandwidth instruction delivery, prediction, and speculation



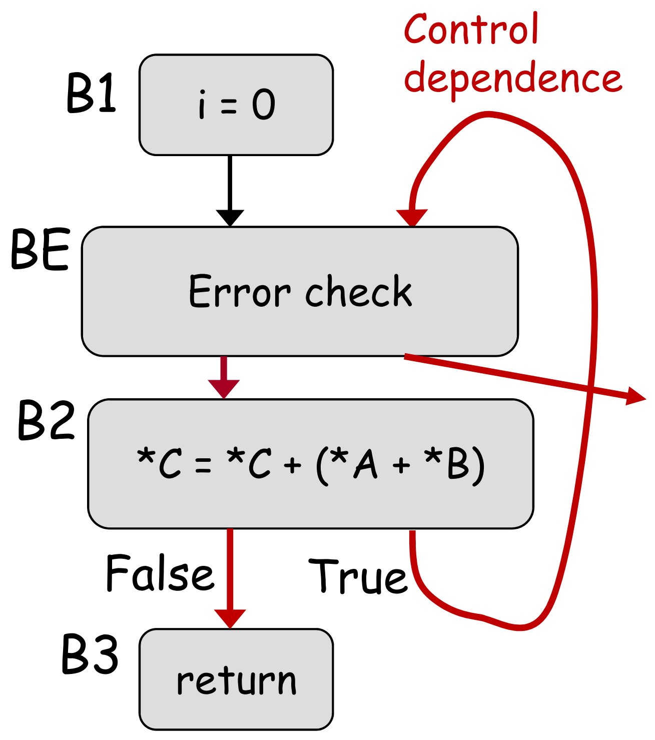
Exploiting Instruction Level parallelism (ILP)

- A superscalar processor has to handle some flows efficiently to exploit ILP
  - Control flow
    - 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, BEQ)
    - Another obstacle is instruction cache
  - Register data flow
    - Dynamic scheduling
  - Memory data flow

Exploiting Instruction Level Parallelism (ILP)

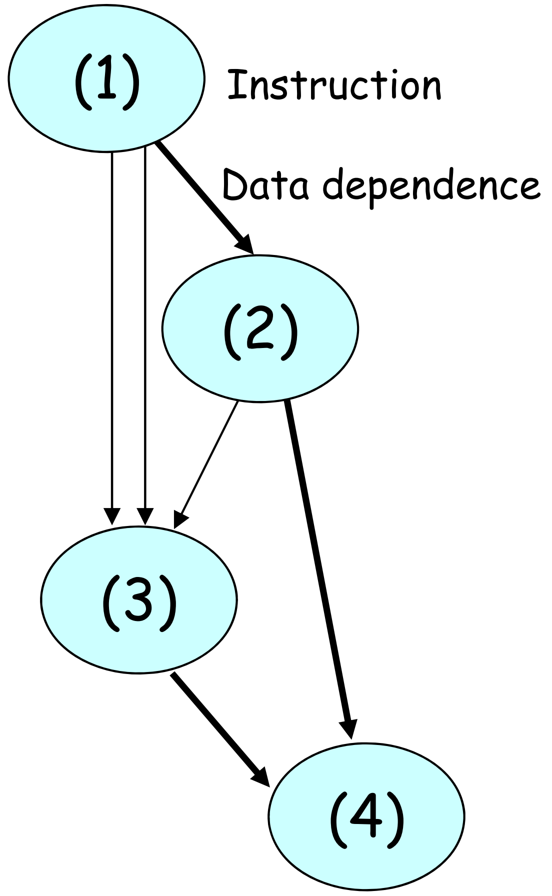
What is the solution?

Prediction & speculation



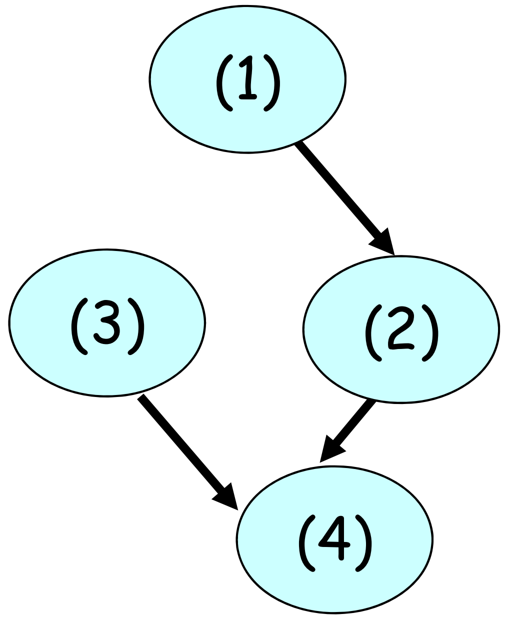
Control flow graph

4 cycles for 4 insns
ILP = 1.0



Data flow graph

3 cycles for 4 insns
ILP = 1.33



Data flow graph

Exercise: what is data dependence

- Draw a data flow graph for each instruction stream

```
R3 = R2 + 1 (1)
R5 = R4 + 2 (2)
R7 = R6 + 3 (3)
```

Instruction stream 1

```
R3 = R2 + 1 (1)
R5 = R4 + 2 (2)
R7 = R3 + 3 (3)
```

Instruction stream 2

```
R3 = R2 + 1 (1)
R3 = R4 + 2 (2)
R7 = R6 + 3 (3)
```

Instruction stream 3

```
R3 = R2 + 1 (1)
R5 = R4 + 2 (2)
R4 = R6 + 3 (3)
```

Instruction stream 4

True data dependence

- Insn i writes a register that insn j reads, RAW (read after write)
- Program order must be preserved to ensure insn j receives the value of insn i.

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

```
Assume R3=10, R5=3
20 = 10 x 2       (1)
21 = 20 + 1       (2)
5  = 3  + 2       (3)
26 = 5  + 21      (4)
```

```
Assume R3=10, R5=3
20 = 10 x 2       (1)
21 = 20 + 1       (2)
41 = 20 + 21      (4)
5  = 3  + 2       (3)
```

Output dependence

- Insn i and j write the same register, WAW (write after write)
- Program order must be preserved to ensure that the value finally written corresponds to instruction j.

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

```
Assume R3=10, R5=3
20 = 10 x 2         (1)
21 = 20 + 1         (2)
5  = 3  + 2         (3)
26 = 5  + 21        (4)
```

```
Assume R3=10, R5=3
5  = 3  + 2         (3)
20 = 10 x 2         (1)
21 = 20 + 1         (2)
41 = 20 + 21        (4)
```

Antidependence

- Insn i reads a register that insn j writes, WAR (write after read)
- Program order must be preserved to ensure that i reads the correct value.

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

```
Assume R3=10, R5=3
20 = 10 x 2       (1)
21 = 20 + 1       (2)
5  = 3  + 2       (3)
26 = 5  + 21      (4)
```

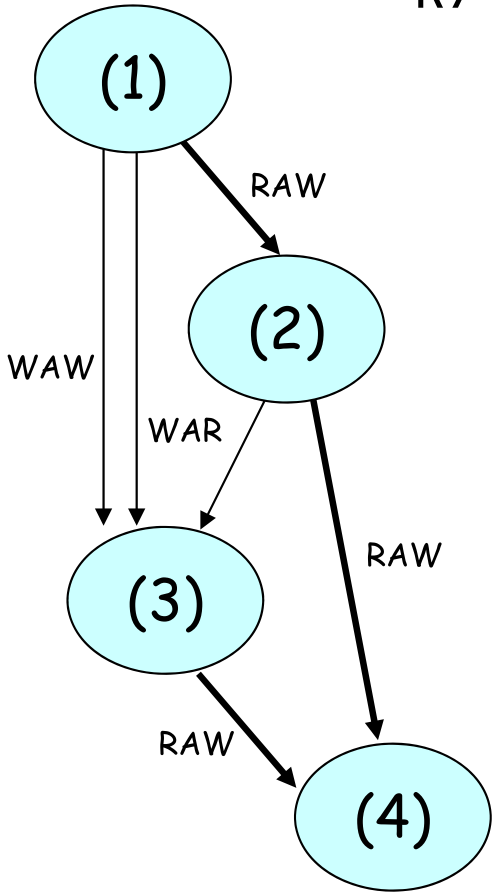
```
Assume R3=10, R5=3
20 = 10 x 2       (1)
5  = 3  + 2       (3)
6  = 5  + 1       (2)
11 = 5  + 6       (4)
```

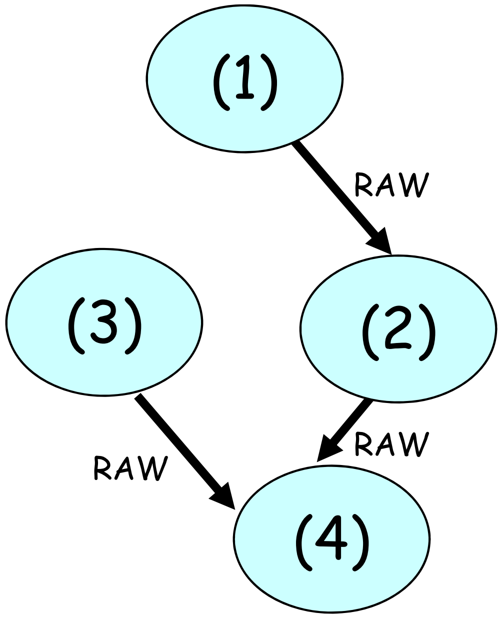
Data dependence and renaming

- True data dependence (RAW)
- Name dependences
  - Output dependence (WAW)
  - Antidependence (WAR)

```
R3 = R3 x R5      (1)
R4 = R3 + 1       (2)
R8 = R5 + 2       (3)
R7 = R8 + R4      (4)
```

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





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

Exercise: register renaming

- Rename the following instruction stream using physical registers of p9, p10, p11, and p12

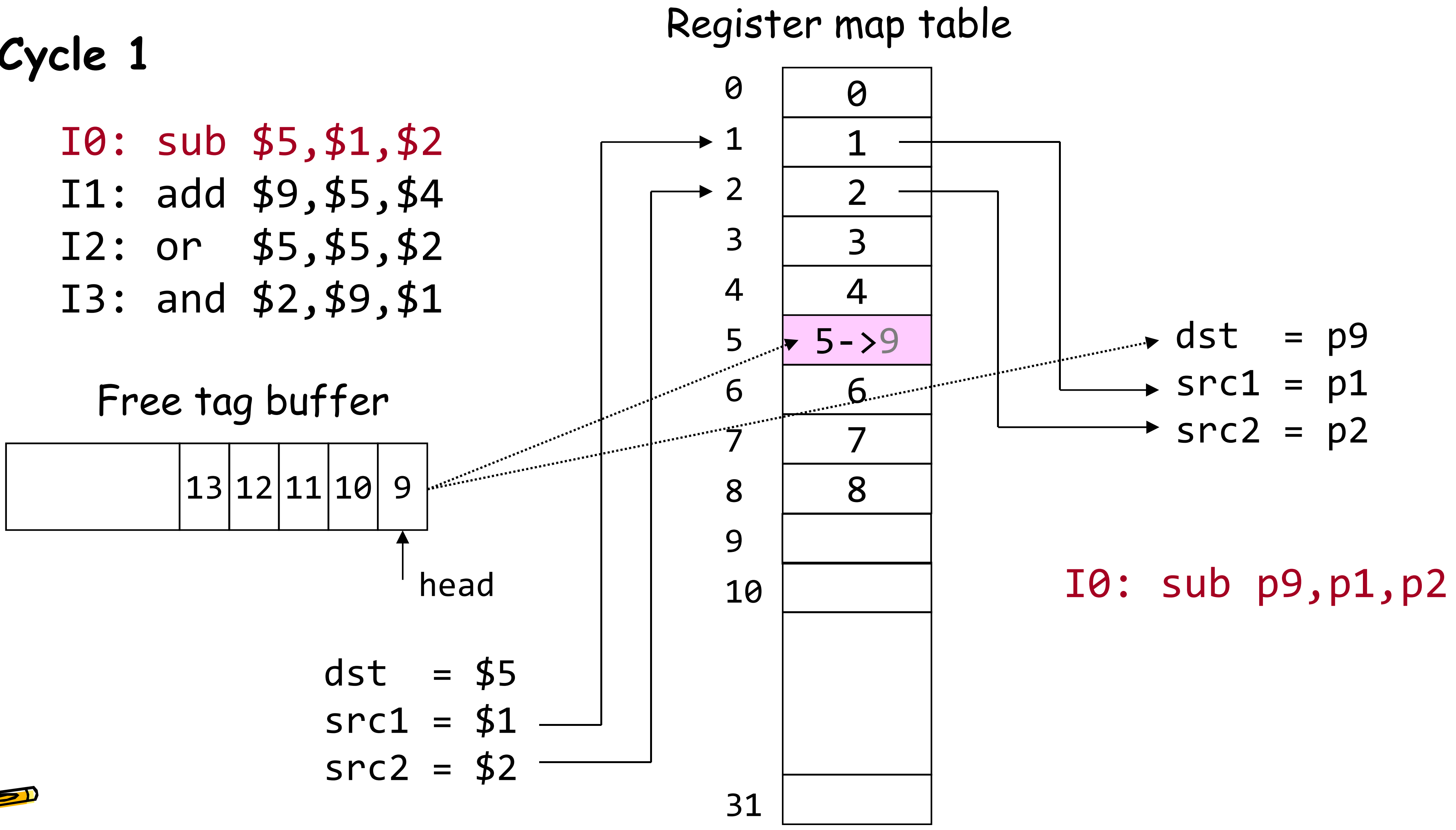
```
I0: sub $5,$1,$2
I1: add $9,$5,$4
I2: or  $5,$5,$2
I3: and $2,$9,$1
```

Example behavior of register renaming (1/4)

- Renaming the first instruction I0

Cycle 1

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



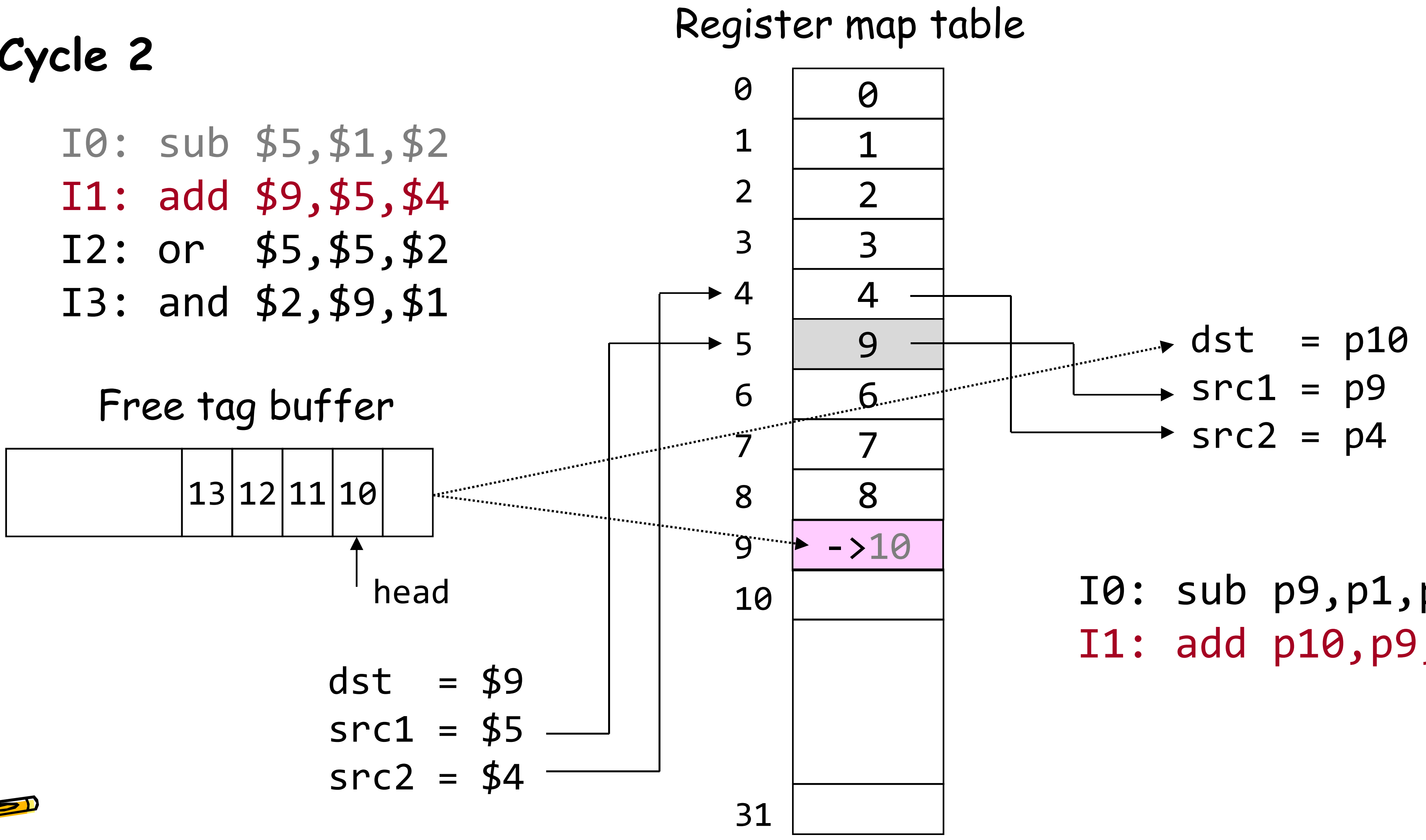
```
I0: sub p9,p1,p2
```

Example behavior of register renaming (2/4)

- Renaming the second instruction I1

Cycle 2

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



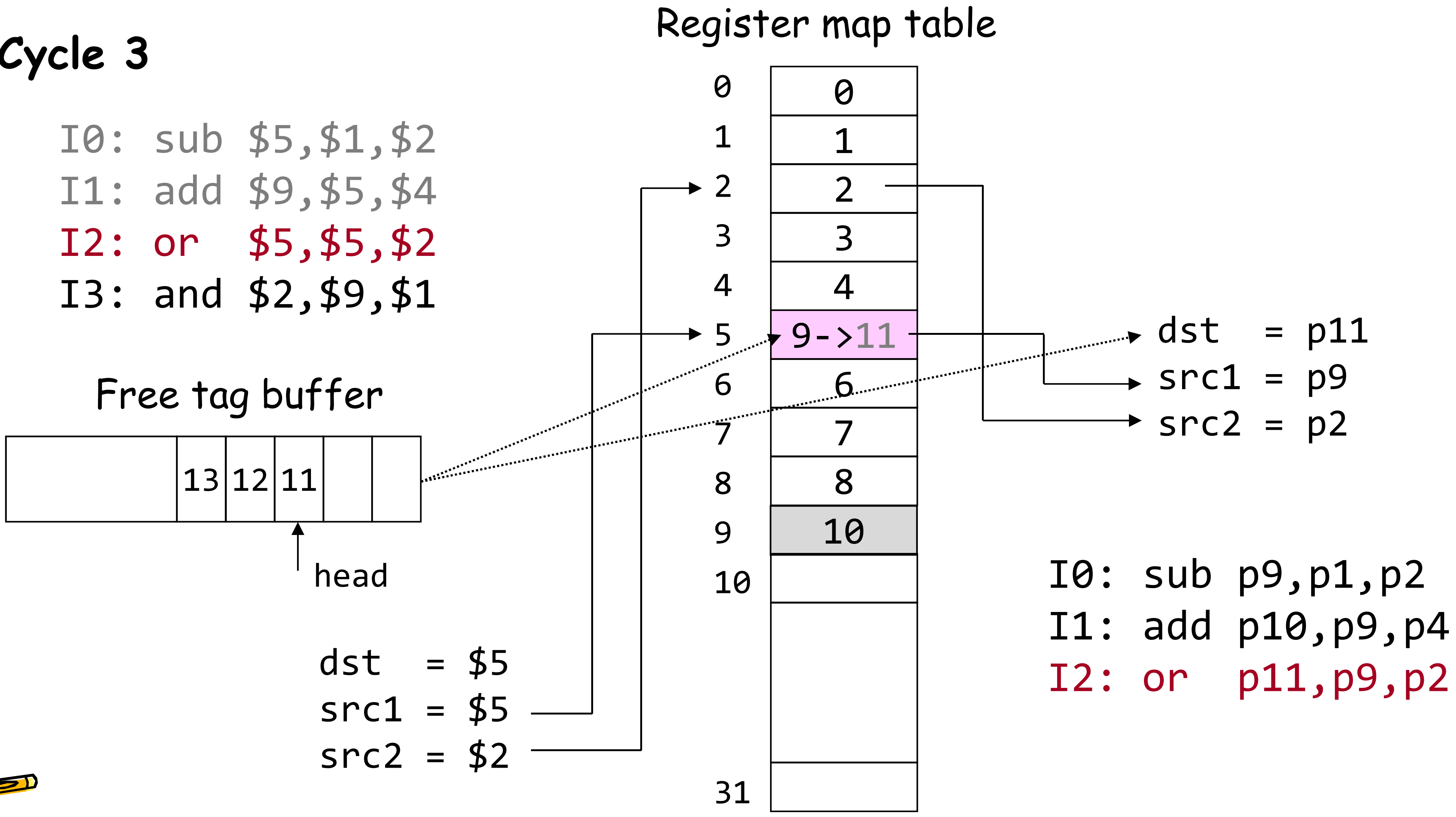
```
I0: sub p9,p1,p2
I1: add p10,p9,p4
```

Example behavior of register renaming (3/4)

- Renaming instruction I2

Cycle 3

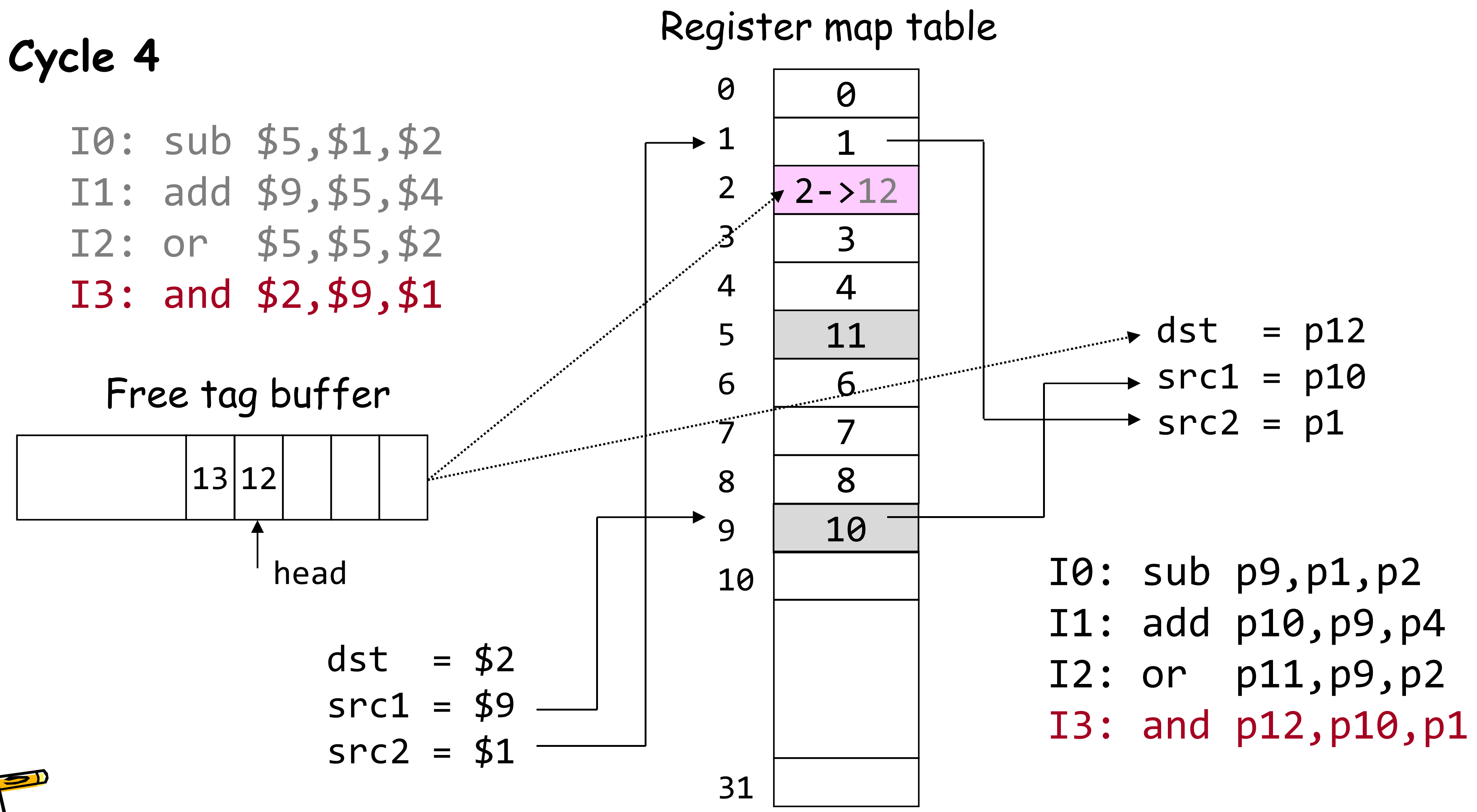
```
I0: sub $5,$1,$2
I1: add $9,$5,$4
I2: or  $5,$5,$2
I3: and $2,$9,$1
```



```
I0: sub p9,p1,p2
I1: add p10,p9,p4
I2: or  p11,p9,p2
```

Example behavior of register renaming (4/4)

- Renaming instruction I3

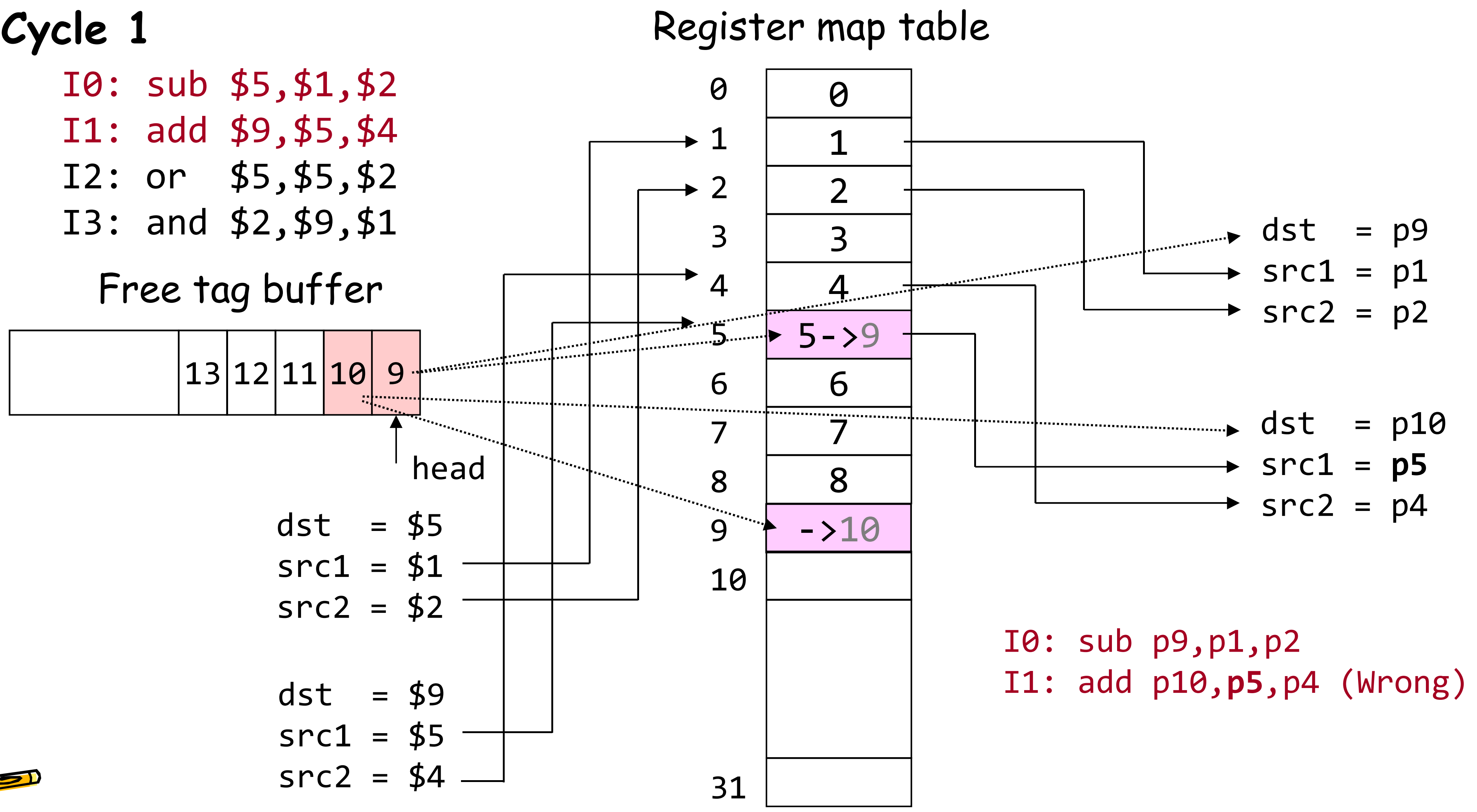


Renaming two instructions per cycle for superscalar

- Renaming instruction I0 and I1

Cycle 1

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



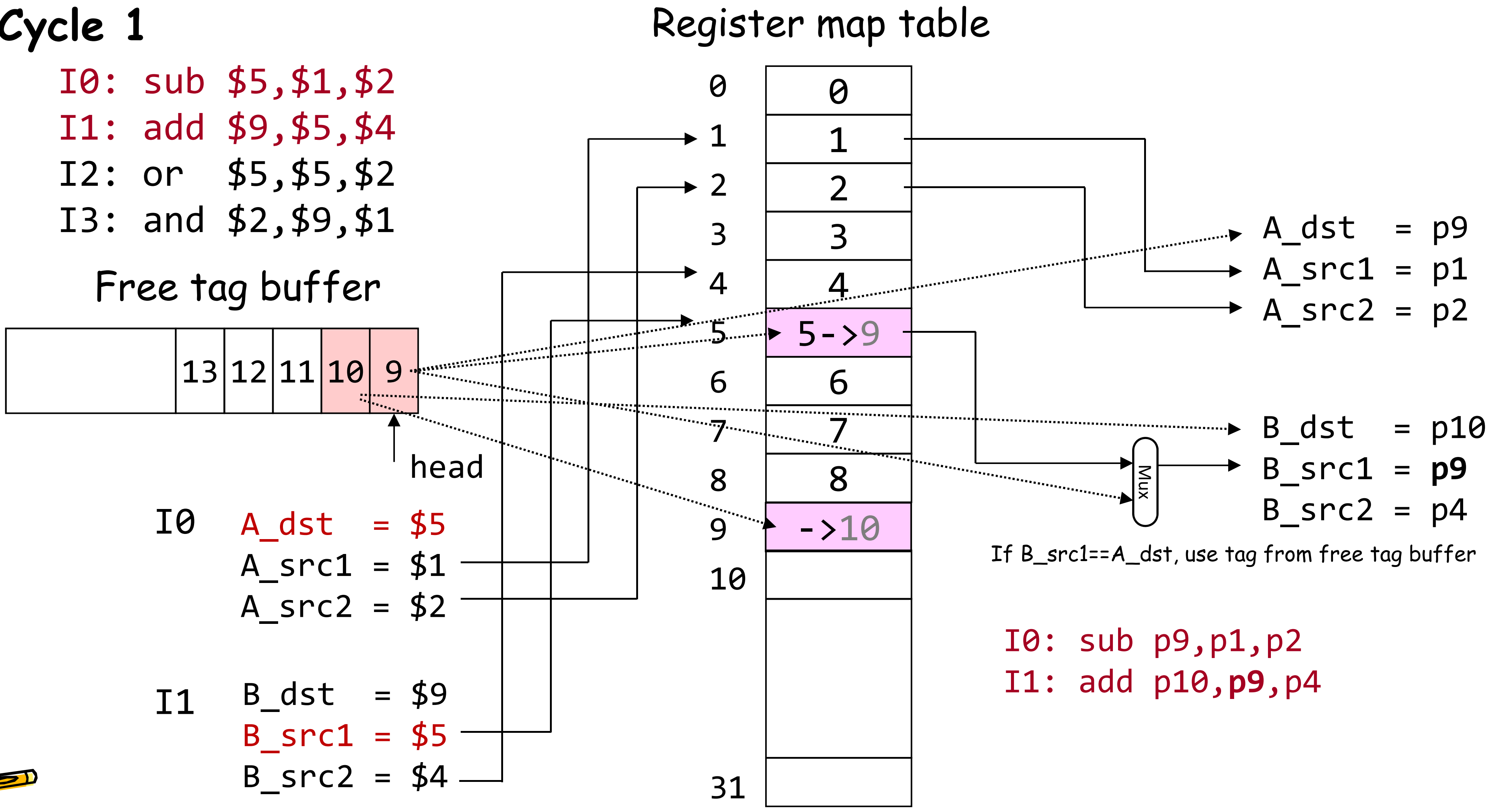
```
I0: sub p9,p1,p2
I1: add p10,p5,p4 (Wrong)
```

Renaming two instructions per cycle for superscalar

- Renaming instruction I0 and I1

Cycle 1

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



If B_src1==A_dst, use tag from free tag buffer

```
I0: sub p9,p1,p2
I1: add p10,p9,p4
```

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