Computer Architecture

Spring 2016

Lecture 10: Out-of-Order Execution & Register Renaming

Shuai Wang

Department of Computer Science and Technology

Nanjing University

In Search of Parallelism

- "Trivial" Parallelism is limited
  - What is trivial parallelism?
    - In-order: sequential instructions do not have dependencies
    - In all previous cases, all instructions executed with or after earlier instructions.
  - Superscalar execution quickly hits a ceiling due to dependency.

- So what is "non-trivial" parallelism? …

Instruction-Level Parallelism (ILP)

ILP is a measure of inter-dependencies between instructions.

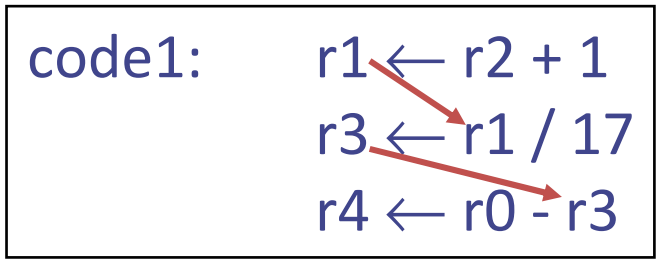
Average ILP = number of instructions / number of cycles required

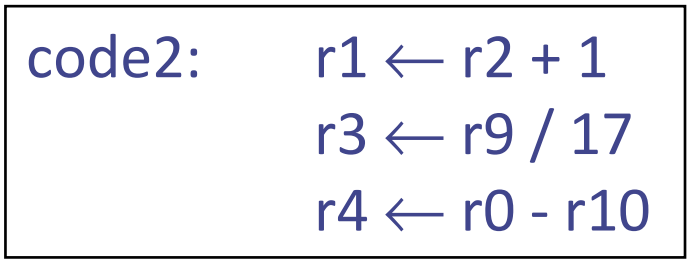
code1: ILP = 1

i.e. must execute serially

code2: ILP = 3

i.e. can execute at the same time





The Problem with In-Order Pipelines

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`addf f0,f1,f2`	F	D	E+	E+	E+	W										
`mulf f2,f3,f2`		F	D	d*	d*	E*	E*	E*	E*	E*	W					
`subf f0,f1,f4`			F	p*	p*	D	E+	E+	E+	W						

- What's happening in cycle 4?
  - **`mulf`** stalls due to **RAW hazard**
    - OK, this is a fundamental problem
  - **`subf`** stalls due to **pipeline hazard**
    - Why? **`subf`** can't proceed into D because **`mulf`** is there
    - That is the only reason, and it isn't a fundamental one
- Why can't **`subf`** go to D in cycle 4 and E+ in cycle 5?

ILP != IPC

- ILP usually assumes
  - Infinite resources
  - Perfect fetch
  - Unit-latency for all instructions
- ILP is a property of the program dataflow

- IPC is the “real” observed metric
  - How many instructions are executed per cycle
- ILP is an upper-bound on the attainable IPC
  - Specific to a particular program

OoO Execution (1/3)

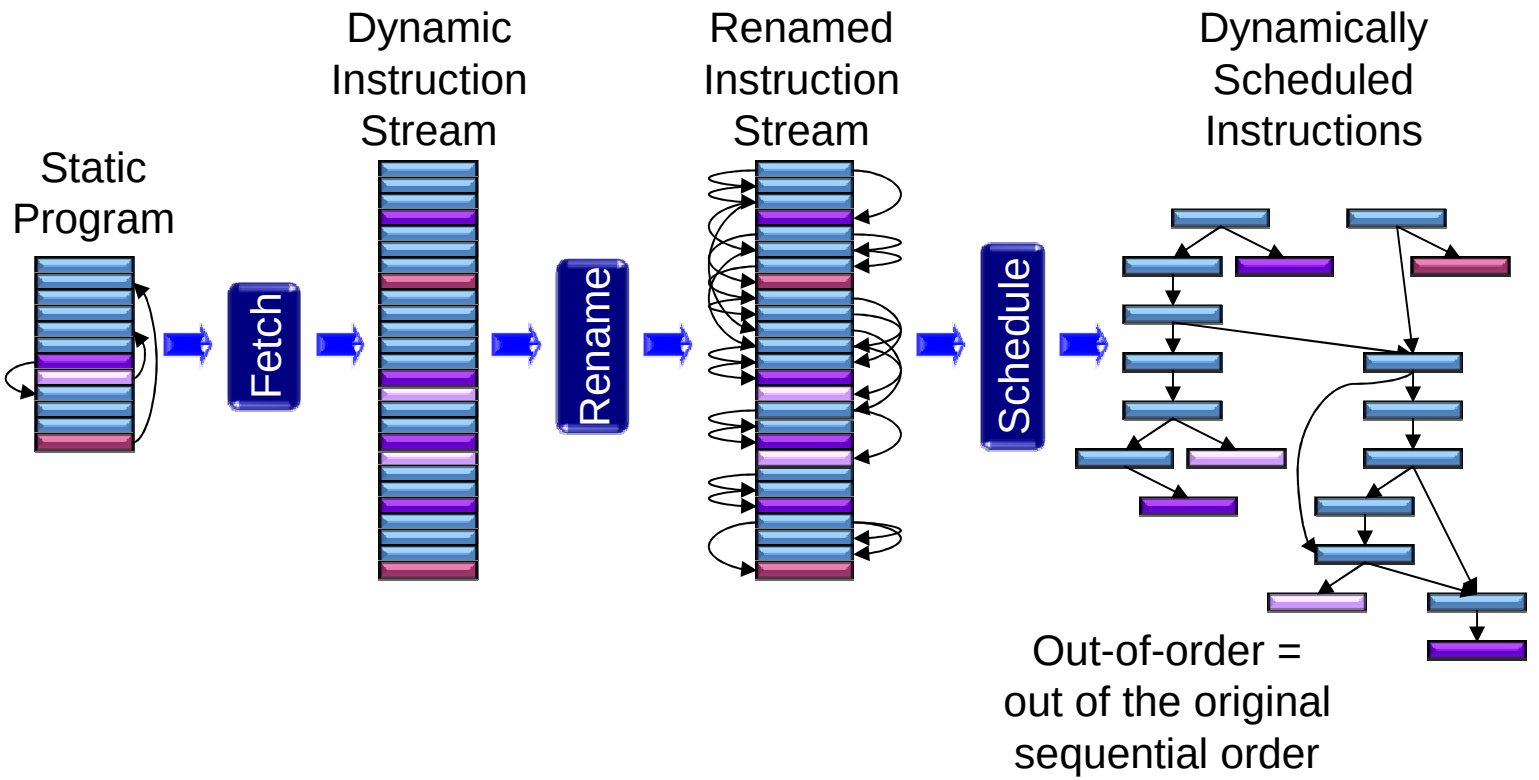
- Dynamic scheduling
  - Totally in the hardware
  - Also called *Out-of-Order execution (OoO)*
- Fetch many instructions into *instruction window*
  - Use branch prediction to speculate past branches
- Rename regs. to avoid false deps. (WAW and WAR)
- Execute insns. as soon as possible
  - As soon as deps. (regs and memory) are known

- Today's machines: 100+ insns. scheduling window

Out-of-Order Execution (2/3)

- Execute insns. in *dataflow* order
  - Often similar but not the same as *program* order
- Use register renaming removes false deps.
- Scheduler identifies when to run insns.
  - Wait for all deps. to be satisfied

Out-of-Order Execution (3/3)



OoO Example (1/2)

A: R1 = R2 + R3

B: R4 = R5 + R6

C: R1 = R1 * R4

D: R7 = LD 0[R1]

E: BEQZ R7, +32

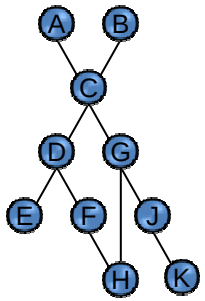
F: R4 = R7 - 3

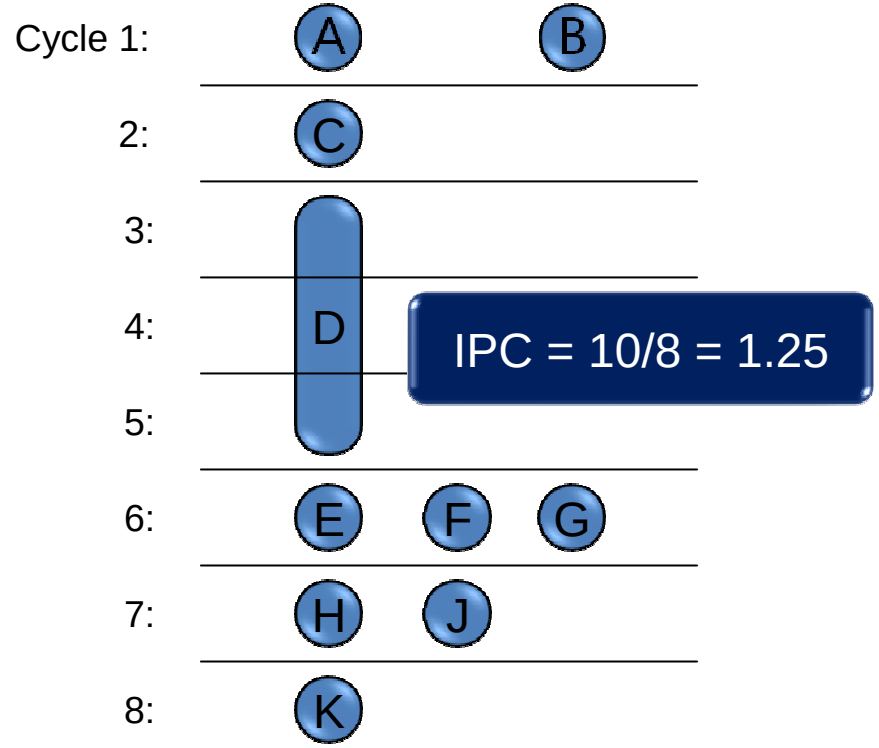
G: R1 = R1 + 1

H: R4 → ST 0[R1]

J: R1 = R1 – 1

K: R3 → ST 0[R1]





OoO Example (2/2)

A: R1 = R2 + R3

B: R4 = R5 + R6

C: R1 = R1 * R4

D: R9 = LD 0[R1]

E: BEQZ R7, +32

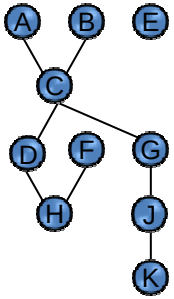
F: R4 = R7 - 3

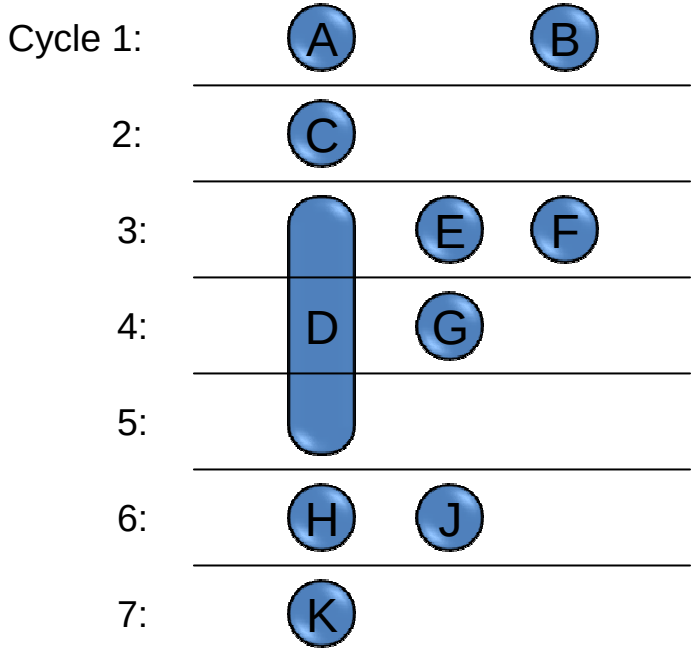
G: R1 = R1 + 1

H: R4 → ST 0[R9]

J: R1 = R1 – 1

K: R3 → ST 0[R1]

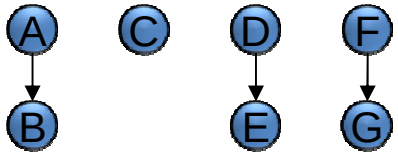


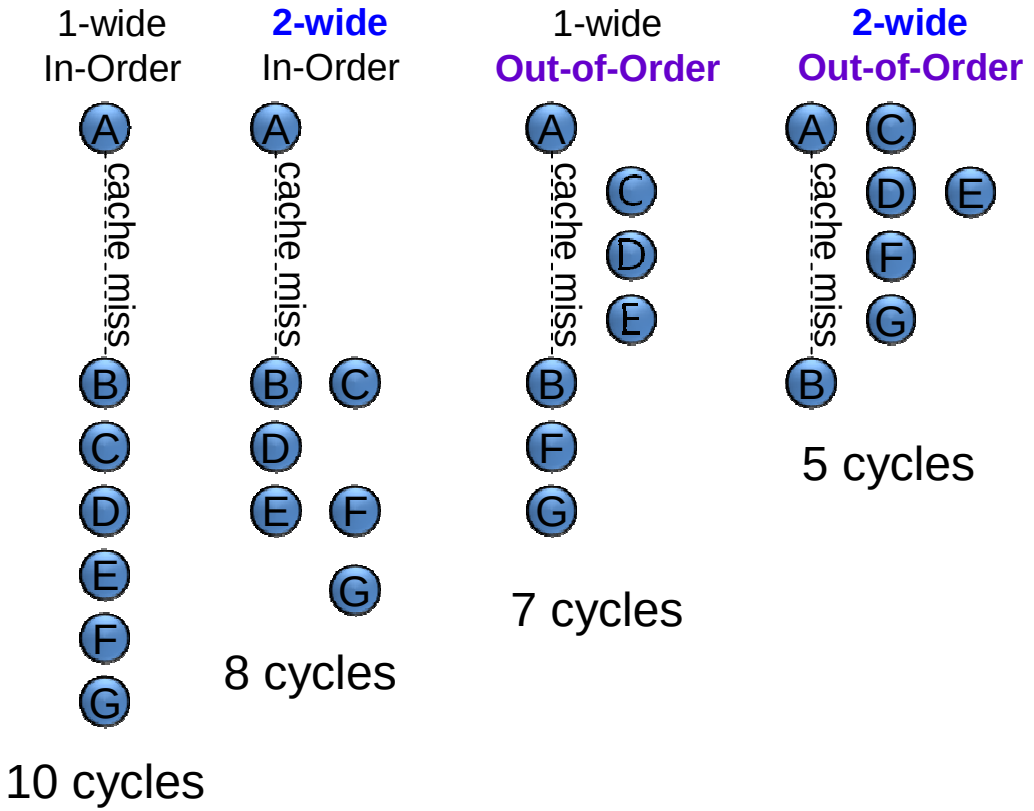


IPC = 10/7 = 1.43

Superscalar != Out-of-Order

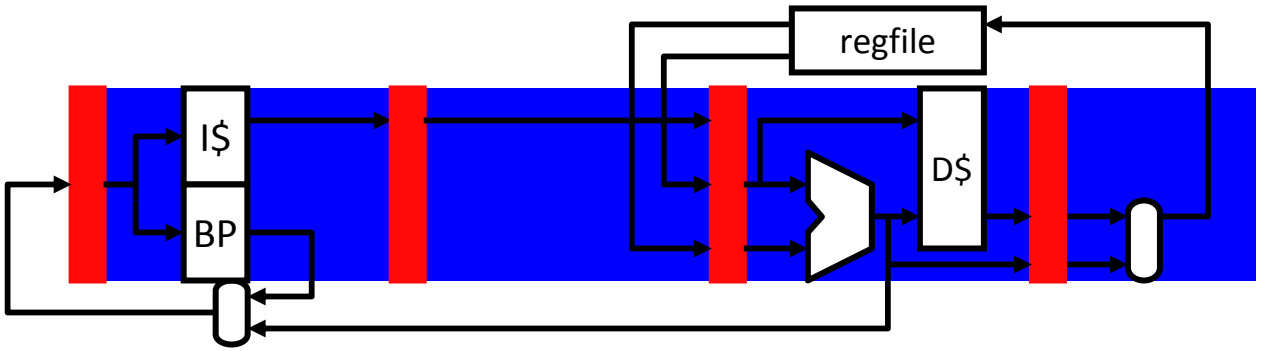
A: R1 = Load 16[R2]
B: R3 = R1 + R4
C: R6 = Load 8[R9]
D: R5 = R2 – 4
E: R7 = Load 20[R5]
F: R4 = R4 – 1
G: BEQ R4, #0



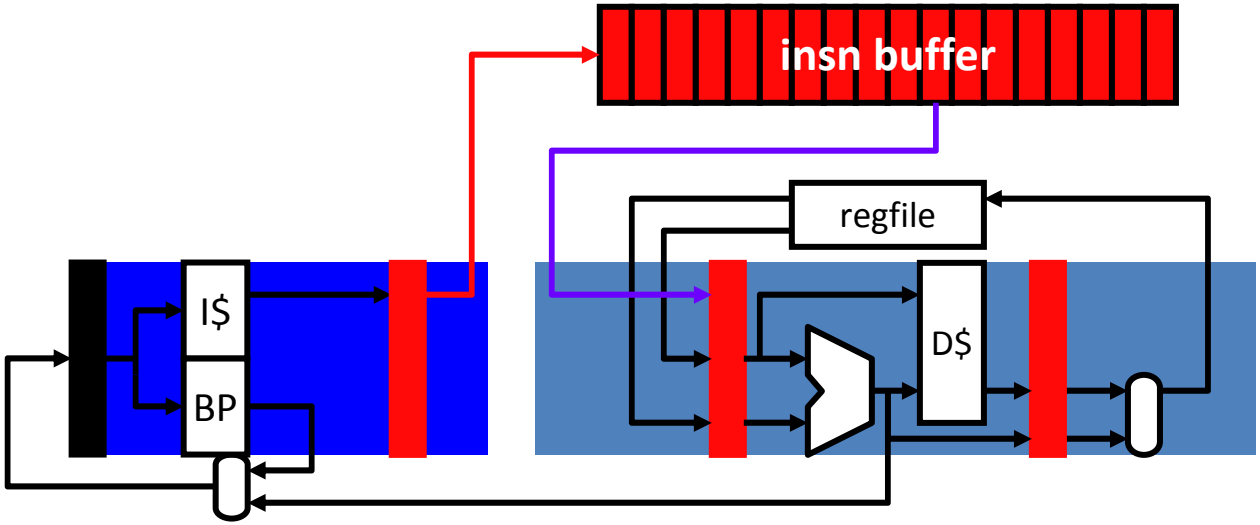


Example Pipeline Terminology

- In-order pipeline
  - F: Fetch
  - D: Decode
  - X: Execute
  - W: Writeback



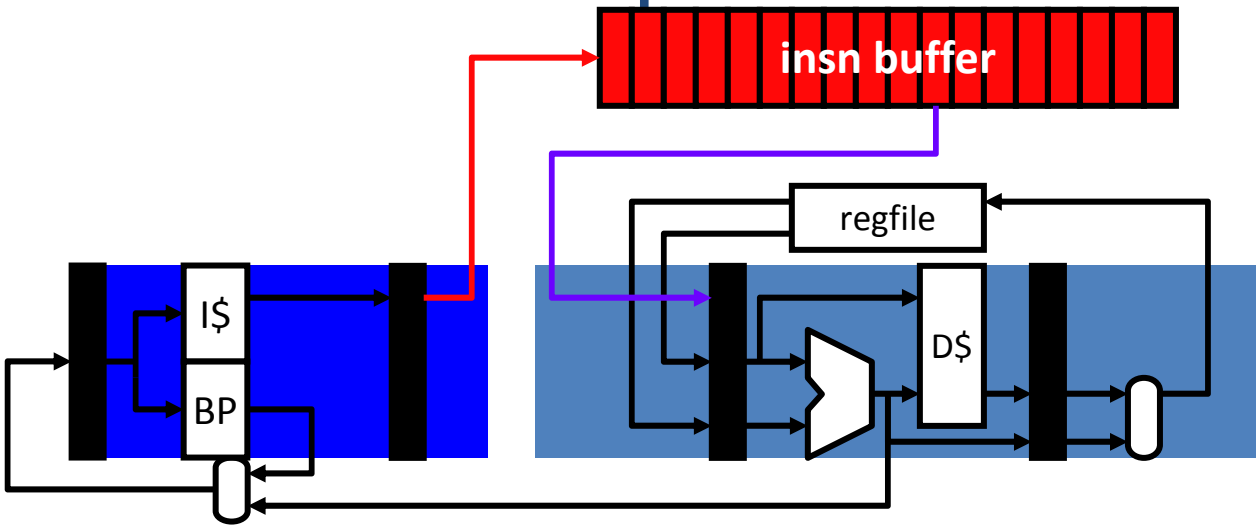
Instruction Buffer



- Trick: ***instruction buffer*** (a.k.a. *instruction window*)
  - A bunch of registers for holding insns.

- Split D into two parts
  - Accumulate decoded insns. in buffer **in-order**
  - Buffer sends insns. down rest of pipeline **out-of-order**

Dispatch and Issue

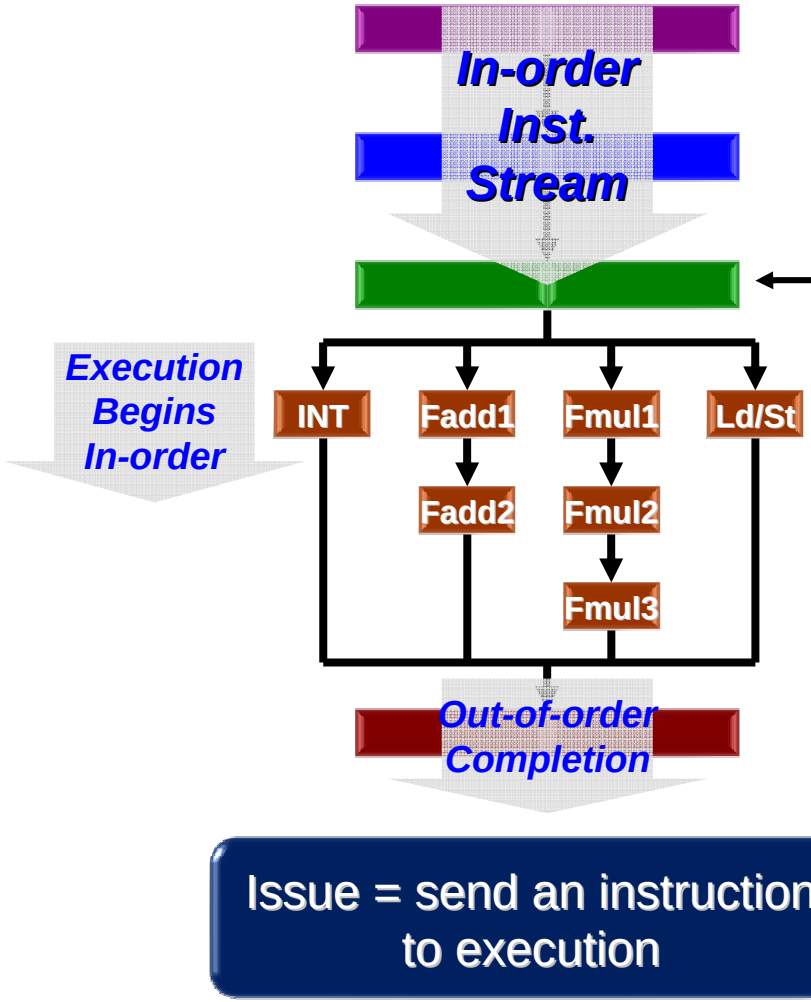


- **Dispatch (D)**: first part of decode
  - Allocate slot in insn. buffer (if buffer is not full)
  - In order: blocks younger insns.
- **Issue (S)**: second part of decode
  - Send insns. from insn. buffer to execution units
  - Out-of-order: doesn't block younger insns.

Our-of-Order Topics

- "*Scoreboarding*"
  - First OoO, no register renaming
- "*Tomasulo's algorithm*"
  - OoO with register renaming
- Handling precise state and speculation
  - P6-style execution (Intel Pentium Pro)
  - R10k-style execution (MIPS R10k)
- Handling memory dependencies

In-Order Issue, OoO Completion



Issue stage needs to check:

1. *Structural Dependence*
2. *RAW Hazard*
3. *WAW Hazard*
4. *WAR Hazard*

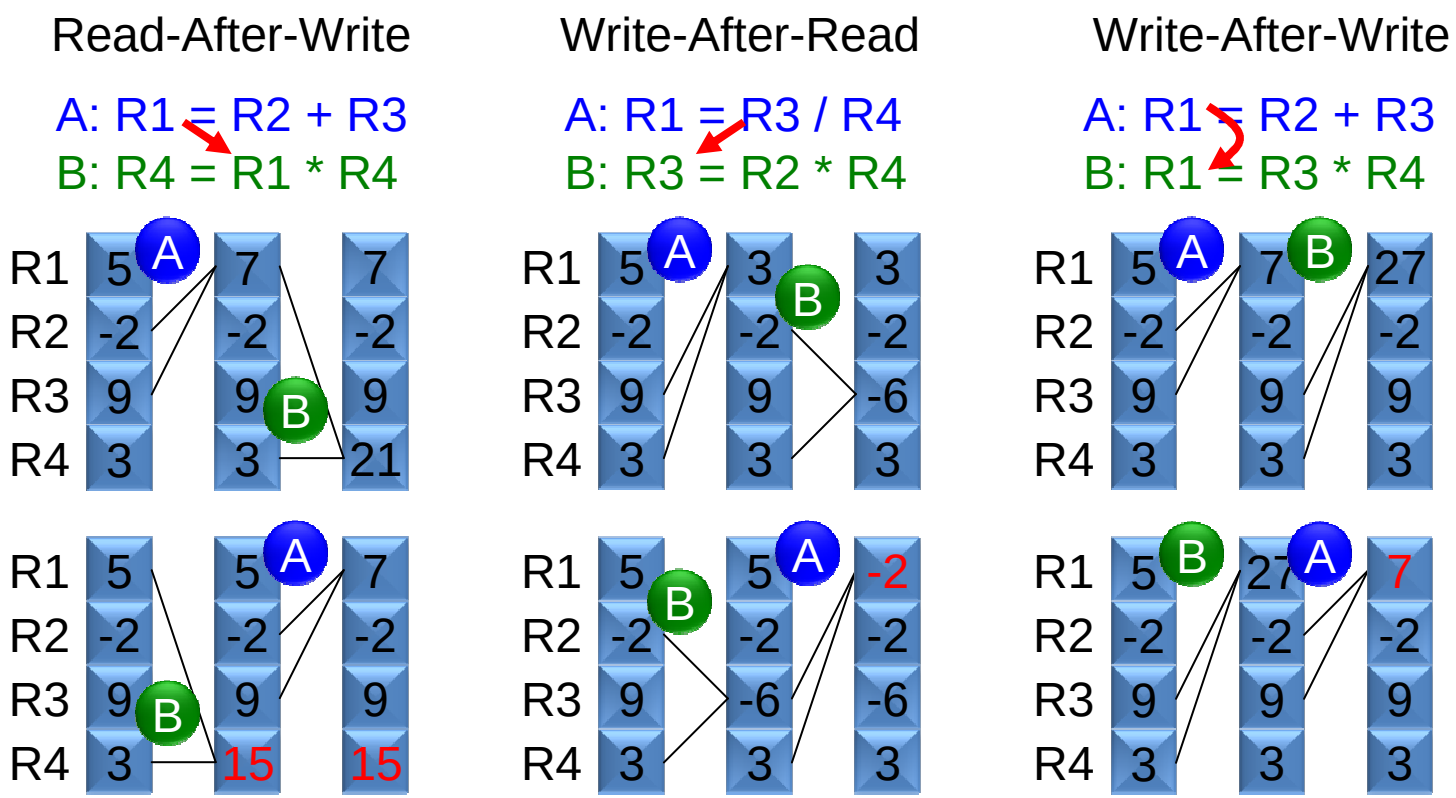
Issue = send an instruction to execution

Track with Simple Scoreboarding

- Scoreboard: a bit-array, 1-bit for each GPR
  - If the bit is *not* set: the register has valid data
  - If the bit is set: the register has stale data

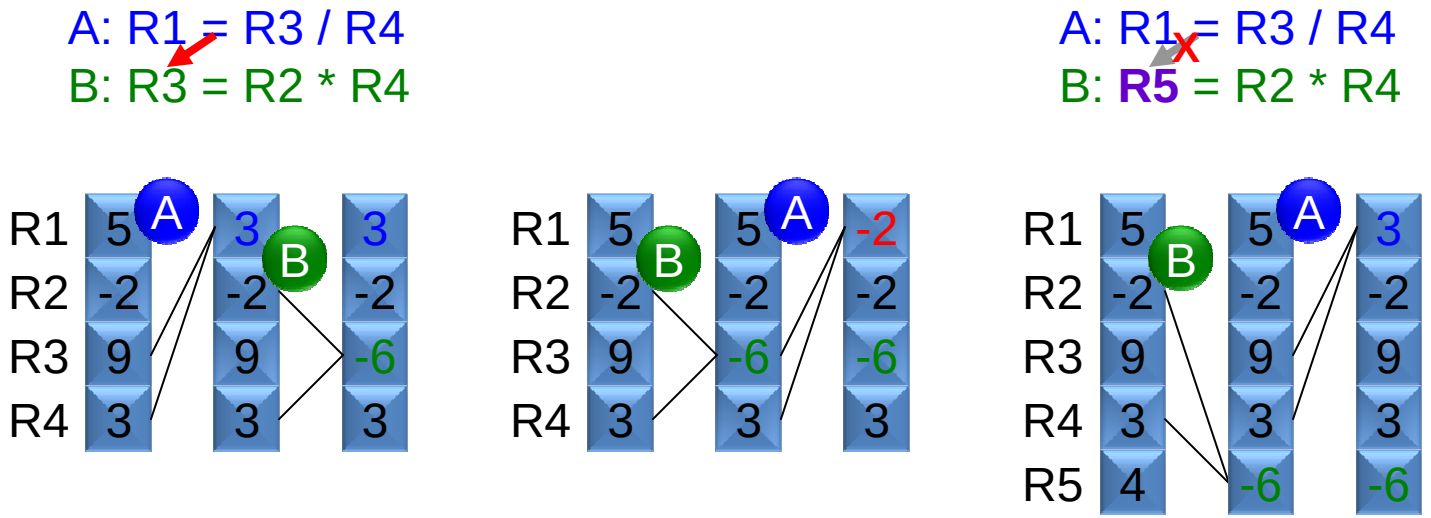
    i.e., some outstanding instruction is going to change it
- Issue in Order: RD ← Fn (RS, RT)
  - If SB[RS] or SB[RT] is set → RAW, stall
  - If SB[RD] is set → WAW, stall
  - Else, dispatch to FU (Fn) and set SB[RD]
- Complete out-of-order
  - Update GPR[RD], clear SB[RD]

Review of Register Dependencies



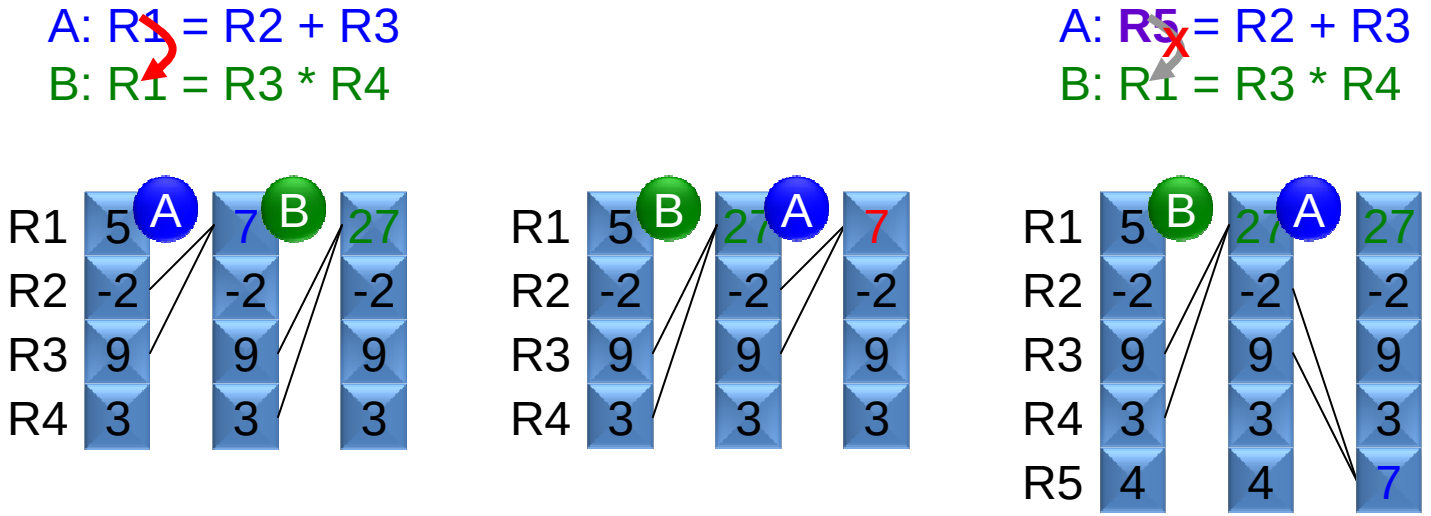
Eliminating WAR Dependencies

- WAR dependencies are from reusing registers



Eliminating WAW Dependencies

- WAW dependencies are also from reusing registers



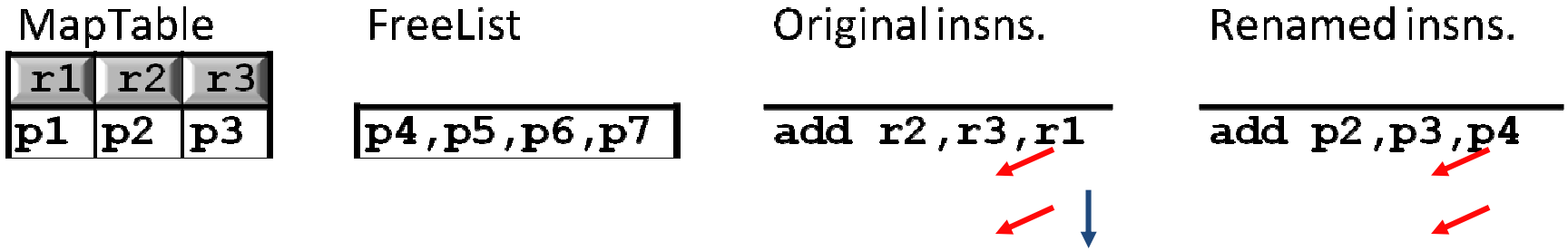
Register Renaming

- *Register renaming* (in hardware)
  - "Change" register names to eliminate WAR/WAW hazards
  - Arch. registers (r1,f0…) are names, not storage locations
  - Can have more locations than names
  - Can have multiple active versions of same name

- How does it work?
  - Map-table: maps names to most recent locations
  - On a write: allocate new location, note in map-table
  - On a read: find location of most recent write via map-table

Register Renaming

- Anti (**WAR**) and output (**WAW**) deps. are false
  - Dep. is on name/location, not on data
  - Given infinite registers, WAR/WAW don't arise
  - Renaming removes WAR/WAW, but leaves **RAW** intact
- Example
  - Names: r1,r2,r3 Locations: p1,p2,p3,p4,p5,p6,p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"



Register Renaming

- Anti (**WAR**) and output (**WAW**) deps. are false
  - Dep. is on name/location, not on data
  - Given infinite registers, WAR/WAW don't arise
  - Renaming removes WAR/WAW, but leaves **RAW** intact
- Example
  - Names: r1,r2,r3 Locations: p1,p2,p3,p4,p5,p6,p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Original insns.

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

Renamed insns.

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```