+

Chapter 16

Instruction-Level Parallelism and Superscalar Processors

William Stallings, Computer Organization and Architecture, 9th Edition

+

Objectives

Parallel execution → High performance

After studying this chapter, you should be able to:

- Explain the difference between superscalar and super pipelined approaches.
- Define instruction-level parallelism.
- Discuss dependencies and resource conflicts as limitations to instruction level parallelism
- Present an overview of the design issues involved in instruction-level parallelism.
- Compare and contrast techniques of improving pipeline performance in RISC machines and superscalar machines.

+
Contents

16.1- Superscalar

Overview

Term first coined in 1987

Refers to a machine that is designed to improve the performance of the execution of scalar instructions

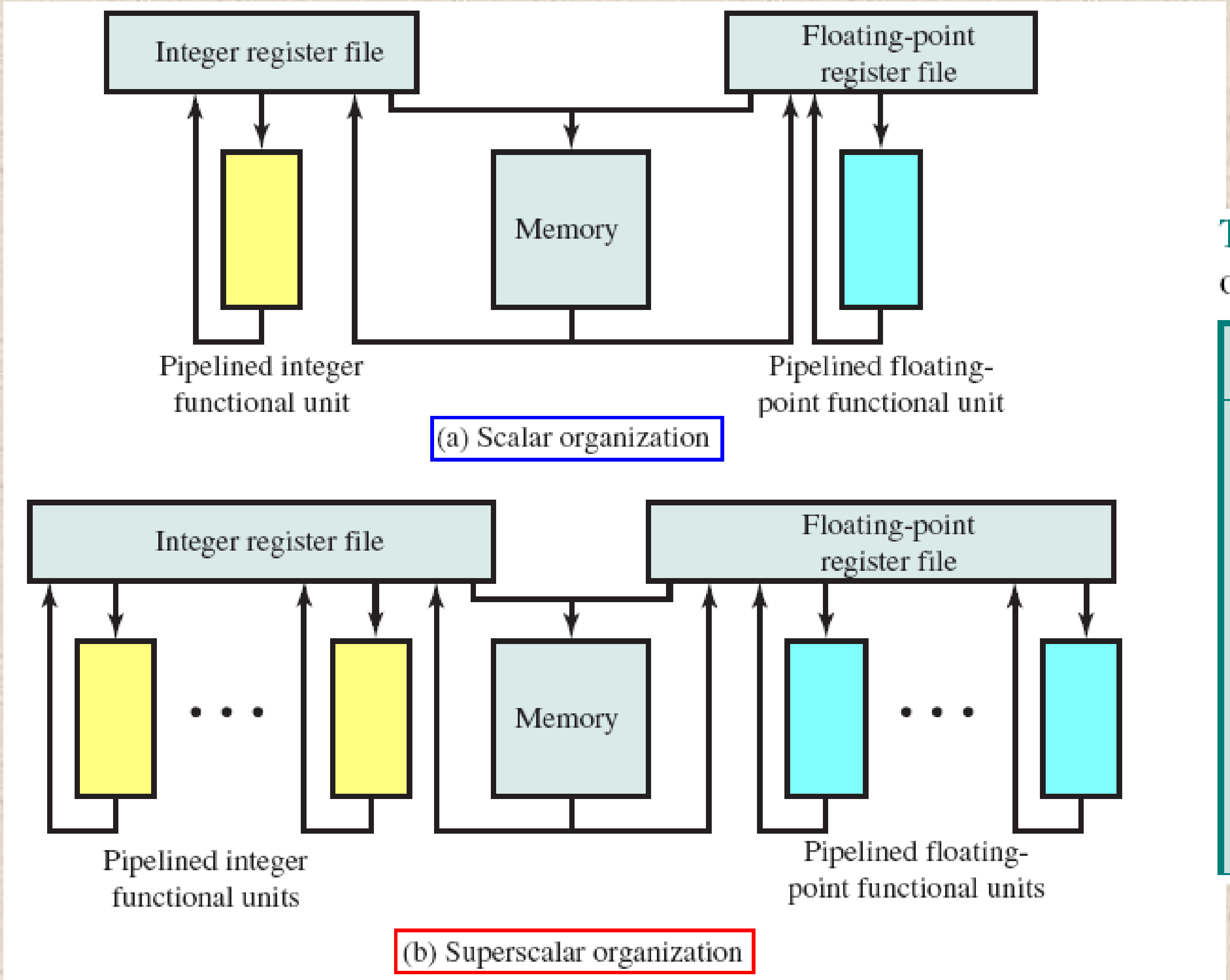
In most applications the bulk (almost) of the operations are on scalar quantities

Represents the next step in the evolution of high-performance general-purpose processors

Essence of the approach is the <u>ability to execute instructions independently and concurrently in different pipelines</u>

Concept can be further exploited by allowing instructions to be executed in an order different from the program order

Compare



Integer register file
Floating-point register file
Memory
Pipelined integer functional unit
Pipelined floating-point functional unit
(a) Scalar organization
Integer register file
Floating-point register file
Memory
Pipelined integer functional units
Pipelined floating-point functional units
(b) Superscalar organization

Figure 16.1 Superscalar Organization Compared to Ordinary Scalar Organization

Some results

Table 16.1 Reported Speedups of Superscalar-Like Machines

Reference	Speedup
[TJAD70]	1.8
[KUCK77]	8
[WEIS84]	1.58
[ACOS86]	2.7
[SOHI90]	1.8
[SMIT89]	2.3
[JOUP89b]	2.2
[LEE91]	7

Comparison of Superscalar and Superpipeline Approaches

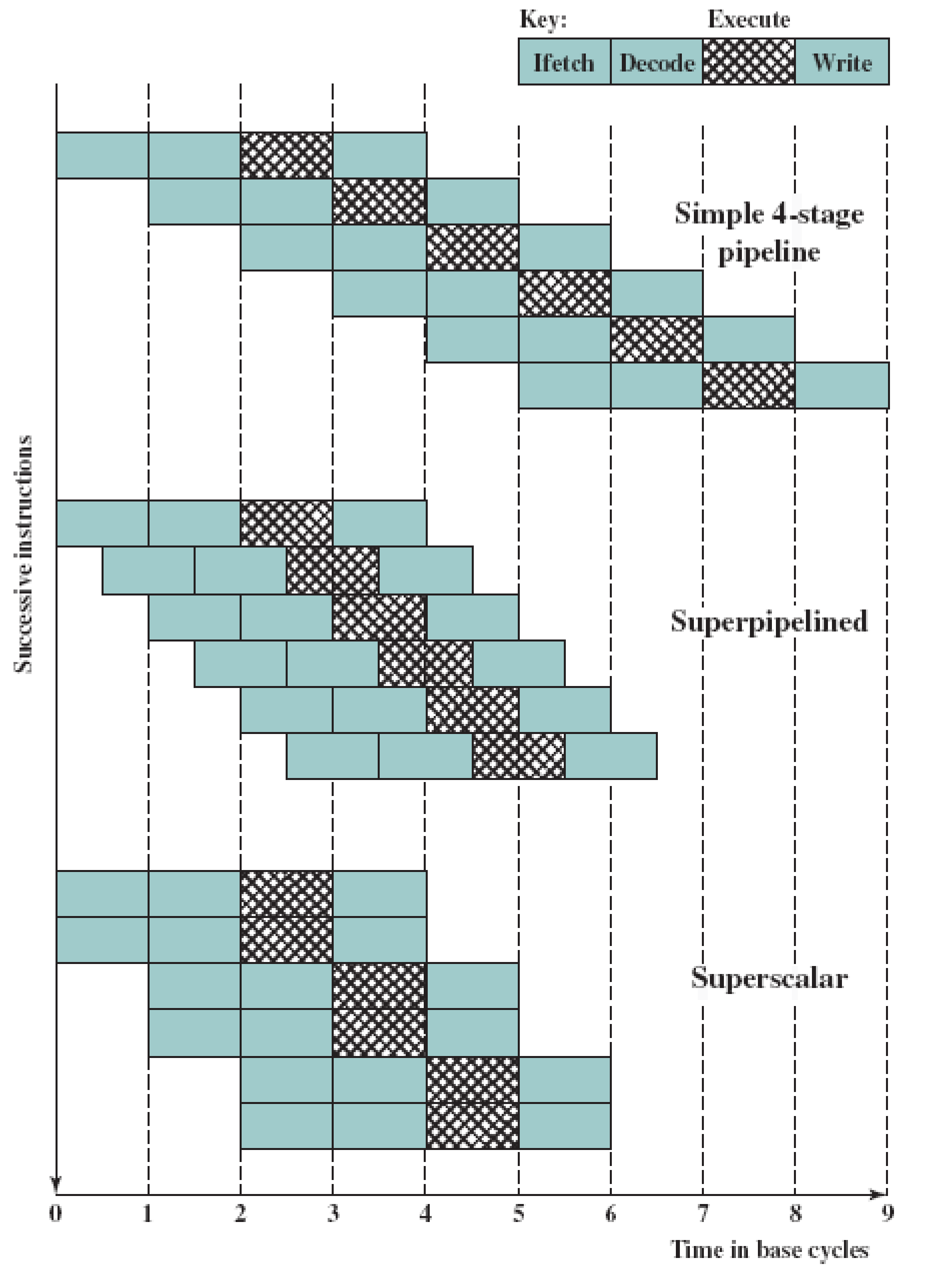


Figure 16.2 Comparison of Superscalar and Superpipeline Approaches

+

Constraints

- Instruction level parallelism
  - Refers to the degree to which the instructions of a program can be executed in parallel
  - A combination of compiler based optimization and hardware techniques can be used to maximize instruction level parallelism
- **Limitations**:
  - True data dependency
  - Procedural dependency
  - Resource conflicts
  - Output dependency
  - Anti-dependency

Input of the next instruction is the output of the previous (RAW)

Previous instruction is a branch, code of the target can cause affects on input of the next

2 instructions access the same resource (bus, registers,…)

2 instructions write values to the same output (Write-after-write - WAW)

Write-after-read situation (WAR)

Situations in which parallel executions can not be used

Constraints - Examples

```
1.   A = 3
2.   B = A
3.   C = B
```

Data dependency → Order of instructions can not be changed → They can not be parallelized

MOV EAX, eff ; /* copy variable eff to the register
MOV EBX, EAX ; /* copy EAX to EBX → Data dependency

```
1.   B = 3
2.   A = B + 1
3.   B = 7
```

Instruction 1, 3 can not be parallelized be cause they are Write-after-write (WAW) → **Output dependency**

```
1.   B = 3
2.   A = B + 1
3.   B = 7
```

Instruction 2 is **anti-dependent** → Order of instructions can not be changed → They can not be parallelized , instruction 3: Write after read (WAR)

Effect of Dependencies

i1 and i2 are executed concurrently

Input of i2 depends on i1 → i2 waits

i2 must be waited due to a branch

i2 waits resources which are being accessed by i1

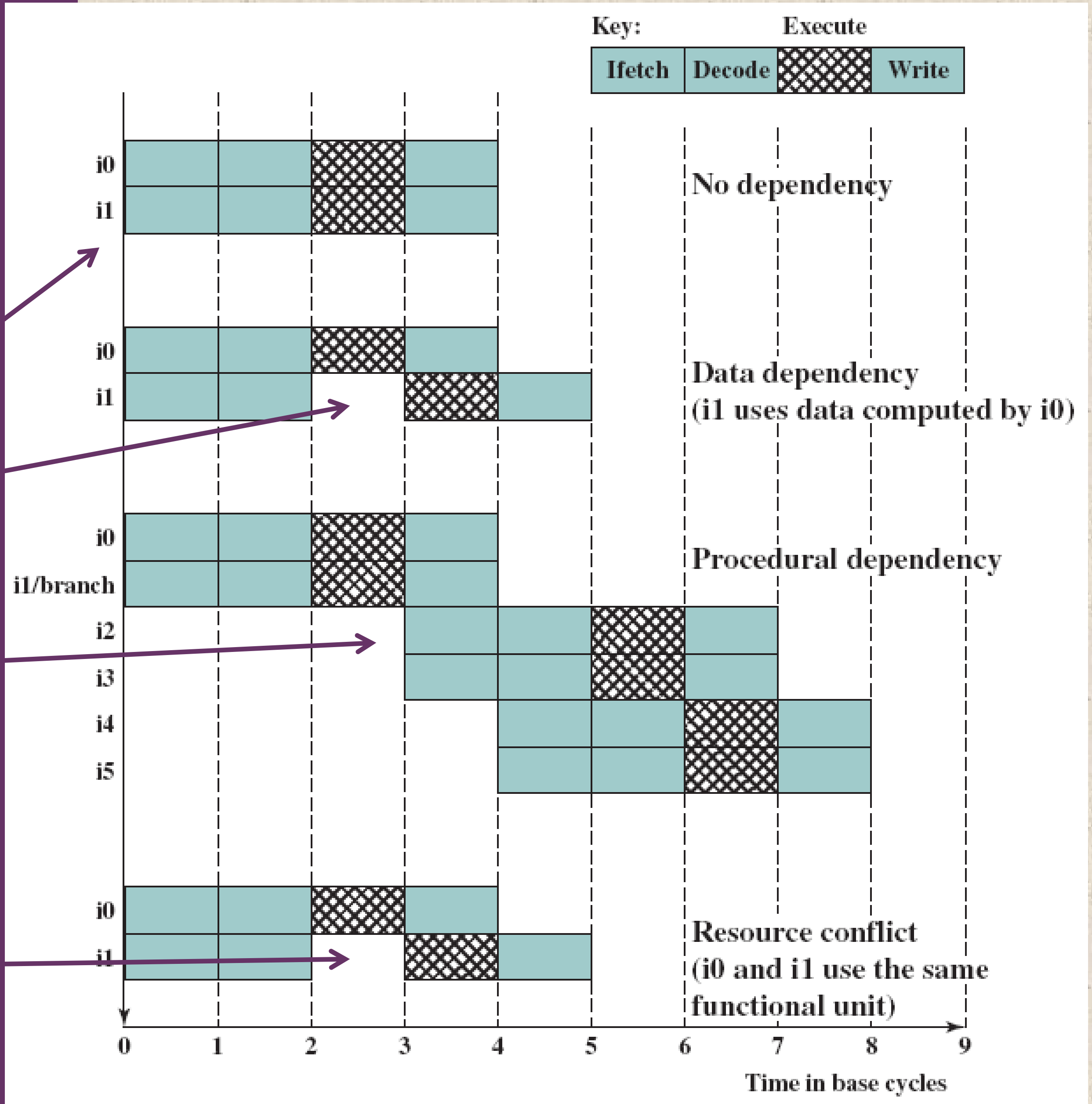


Figure 16.3 Effect of Dependencies

+

Design Issues

Instruction-Level Parallelism and Machine Parallelism

- **Instruction level parallelism**
  - **Instructions** in a sequence are **independent**
  - **Execution** can be **overlapped**
  - Governed by data and procedural dependency
- **Machine Parallelism**
  - Ability to take advantage of instruction level parallelism
  - Governed by number of parallel pipelines

+

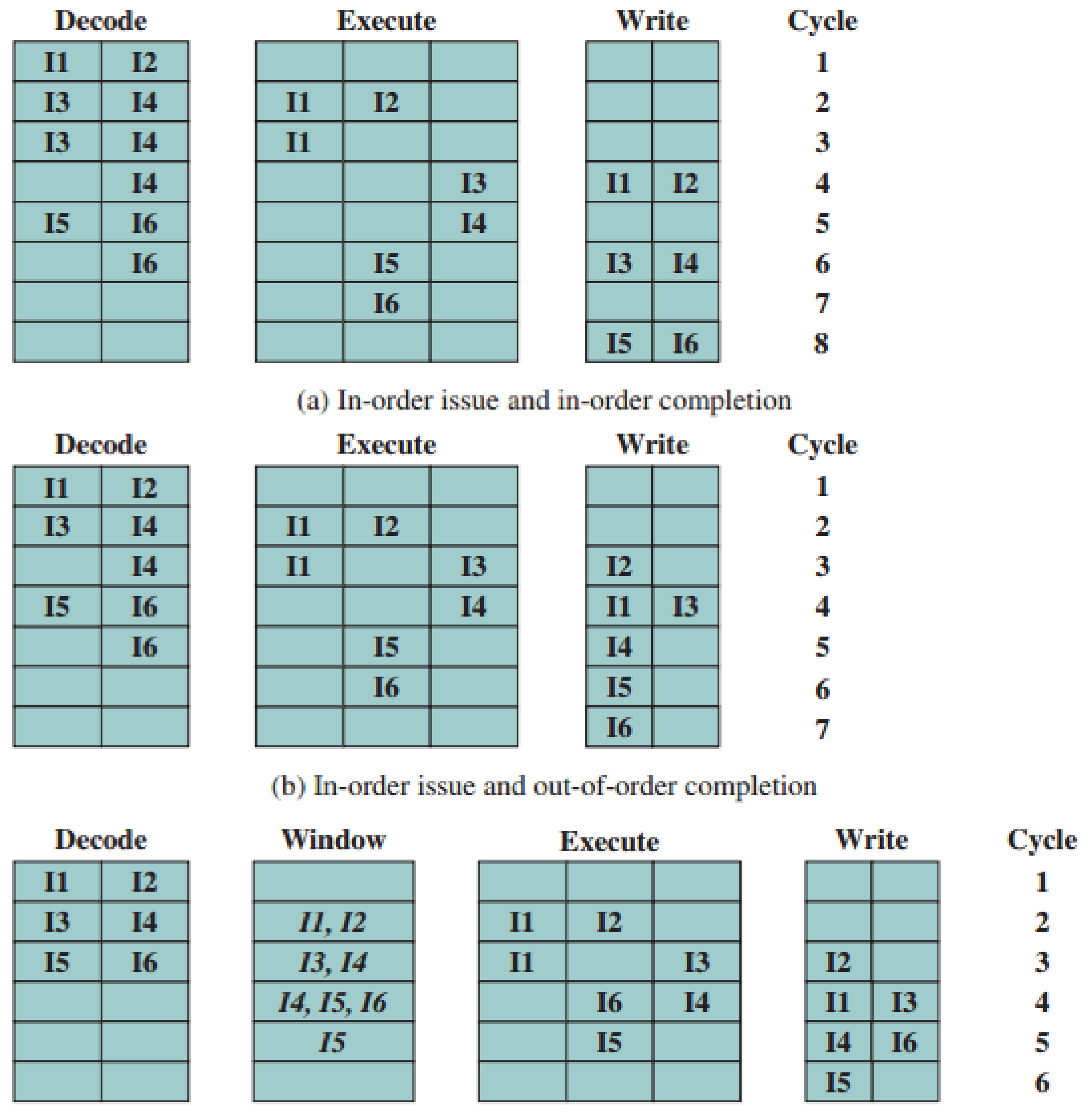
Instruction Issue Policy

Chiến lược phát lệnh

- **Instruction issue**
  - Refers to the process of **initiating** instruction execution **in** the **processor's functional units**
- **Instruction issue policy**
  - Refers to the protocol used to issue instructions
  - Instruction issue occurs when instruction moves from the **decode stage of the pipeline to the first execute stage** of the pipeline
- **Three types of orderings are important:**
  - The **order** in which instructions are **fetched**
  - The **order** in which instructions are **executed**
  - The **order** in which instructions **update** the contents of register and memory locations
- **Superscalar instruction issue policies can be grouped into the following categories:**
  - **In-order** issue with **in-order** completion
  - **In-order** issue with **out-of-order** completion
  - **Out-of-order** issue with **out-of-order** completion

+

Superscalar Instruction Issue and Completion Policies



Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

(a) In-order issue and in-order completion

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

(b) In-order issue and out-of-order completion

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1, I2*	I1	I2				2
I5	I6	*I3, I4*	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

(c) Out-of-order issue and out-of-order completion

Figure 16.4 Superscalar Instruction Issue and Completion Policies

Organization for Out-of-Order Issue with Out-of-Order Completion

An instruction buffer (**instruction window**) is used to store instructions which are ready for executing. After a processor has finished decoding an instruction, it is placed in it. As long as this buffer is not full, the processor can continue to fetch and decode new instructions.

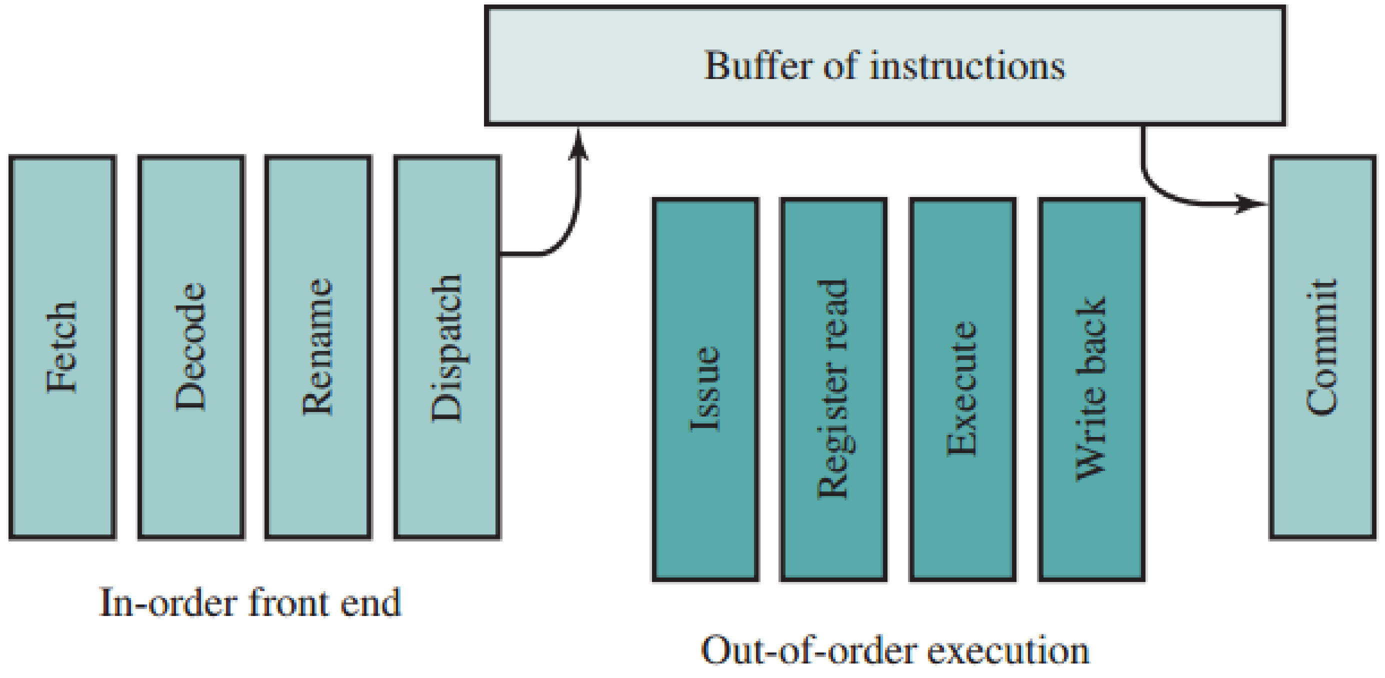


Figure 16.5 Organization for Out-of-Order Issue with Out-of-Order Completion

Any instruction in the buffer will be issued out-of-order if
(1) It needs the particular functional unit that is available, and
(2) No conflicts or dependencies block this instruction.

Another buffer (reorder buffer) can be used as a temporary storage for results completed out of order that are then committed to the register file in program order

Register Renaming

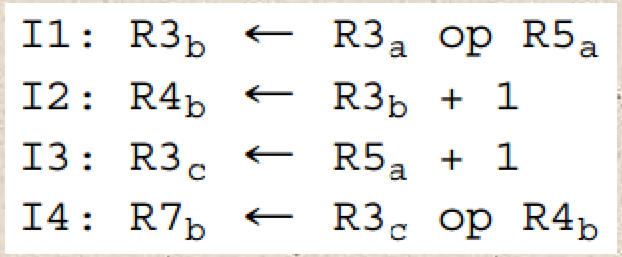
Output and antidependencies occur because register contents may not reflect the correct ordering from the program

May result in a pipeline stall (nghẽn)

Registers allocated dynamically

Compiler techniques attempt to maximize the use of registers → maximizing the number of storage conflicts if parallel execution is applied. Register renaming is a technique of duplication of resources (**more registers are added**). Registers are allocated dynamically by the processor hardware, and they are associated with the values needed by instructions at various points in time. Thus, the same original register reference in several different instructions may refer to different actual registers.

Register Renaming- Example



I1: $R3_b \leftarrow R3_a \text{ op } R5_a$

I2: $R4_b \leftarrow R3_b + 1$

I3: $R3_c \leftarrow R5_a + 1$

I4: $R7_b \leftarrow R3_c \text{ op } R4_b$

R3: logical register

$R3_a$:a hardware register allocated dynamically

When a new allocation is made for a particular logical register, subsequent instruction references to that logical register as a source operand are made to refer to the most recently allocated hardware register (recent in terms of the program sequence of instructions). In this example, the creation of register $R3_c$ in instruction I3 avoids the WAR dependency on the second instruction and the WAW on the first instruction, and it does not interfere with the correct value being accessed by I4. The result is that I3 can be issued immediately; without renaming, I3 cannot be issued until the first instruction is complete and the second instruction is issued.

Machine Parallelism

3 hardware techniques that can be used in a superscalar processor to enhance performance:

(1) Duplication of resources,

(2) Out-of-order issue,

(3) Renaming registers.

Figure 16.6 (next slide) shows the results in mean speedup of the superscalar machine over the scalar machine (without procedural dependencies).

base: processor organization does not duplicate any of the functional units, but it can issue instructions out of order.

+ld/st: duplicates the **lo**a**d**/**st**ore functional unit that accesses a data cache.

+alu: duplicates the ALU,

+both: duplicates both load/store and ALU.

Speedups of Various Machine Organizations Without Procedural Dependencies

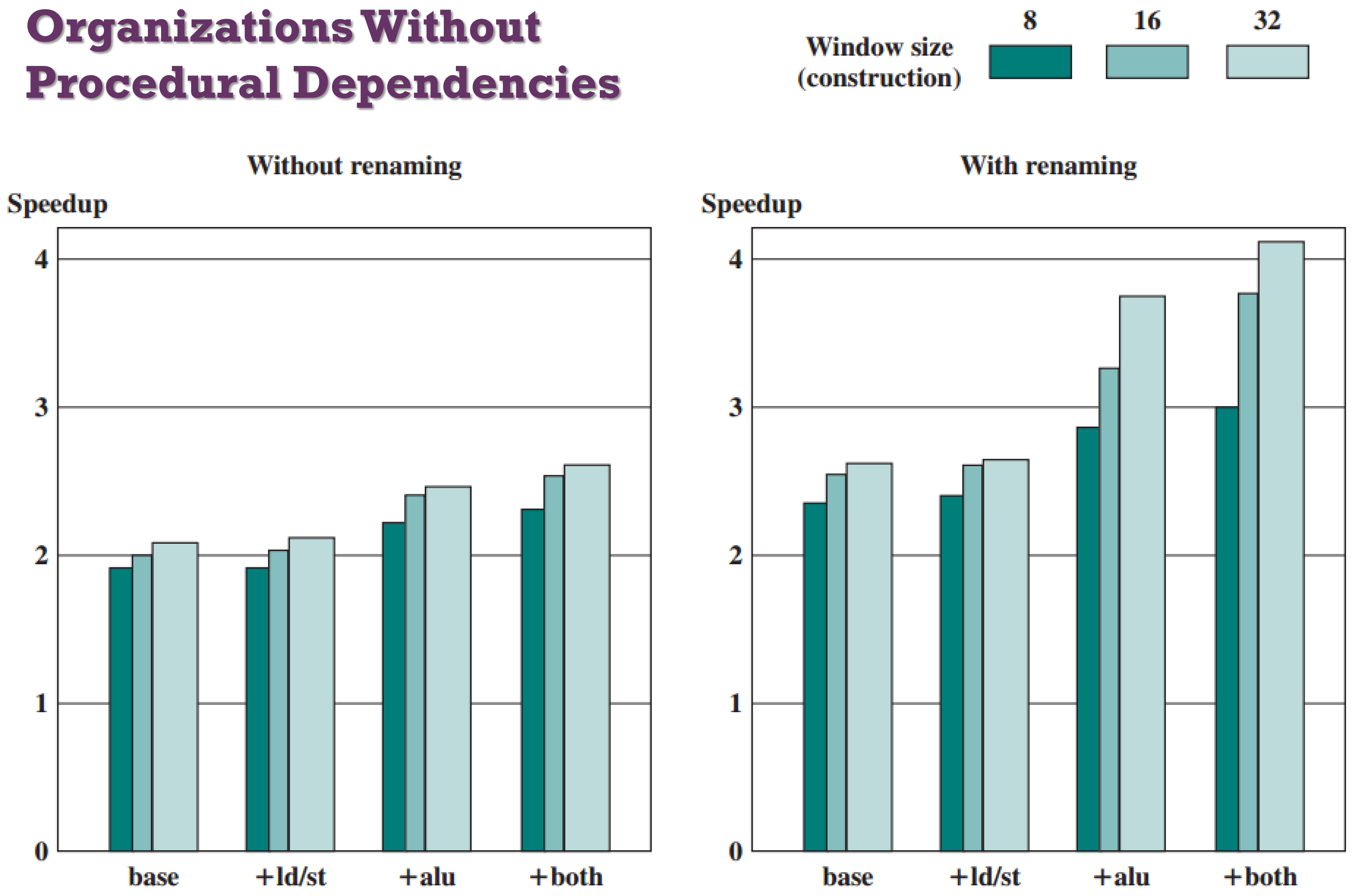


Figure 16.6 Speedups of Various Machine Organizations without Procedural Dependencies

+

Branch Prediction

- Any high-performance pipelined machine must address the issue of dealing with branches
- Intel 80486 addressed the problem by fetching both the next sequential instruction after a branch and speculatively fetching the branch target instruction
- **RISC machines**:
  - **Delayed branch** strategy was explored
  - Processor always executes the single instruction that immediately follows the branch
  - Keeps the pipeline full while the processor fetches a new instruction stream
- **Superscalar machines**:
  - Delayed branch strategy has **less appeal** (không là yêu cầu)
  - Have returned to pre-RISC techniques of branch prediction

Reasons: multiple instructions need to execute in the delay slot, instruction dependencies are major interest

Superscalar Execution

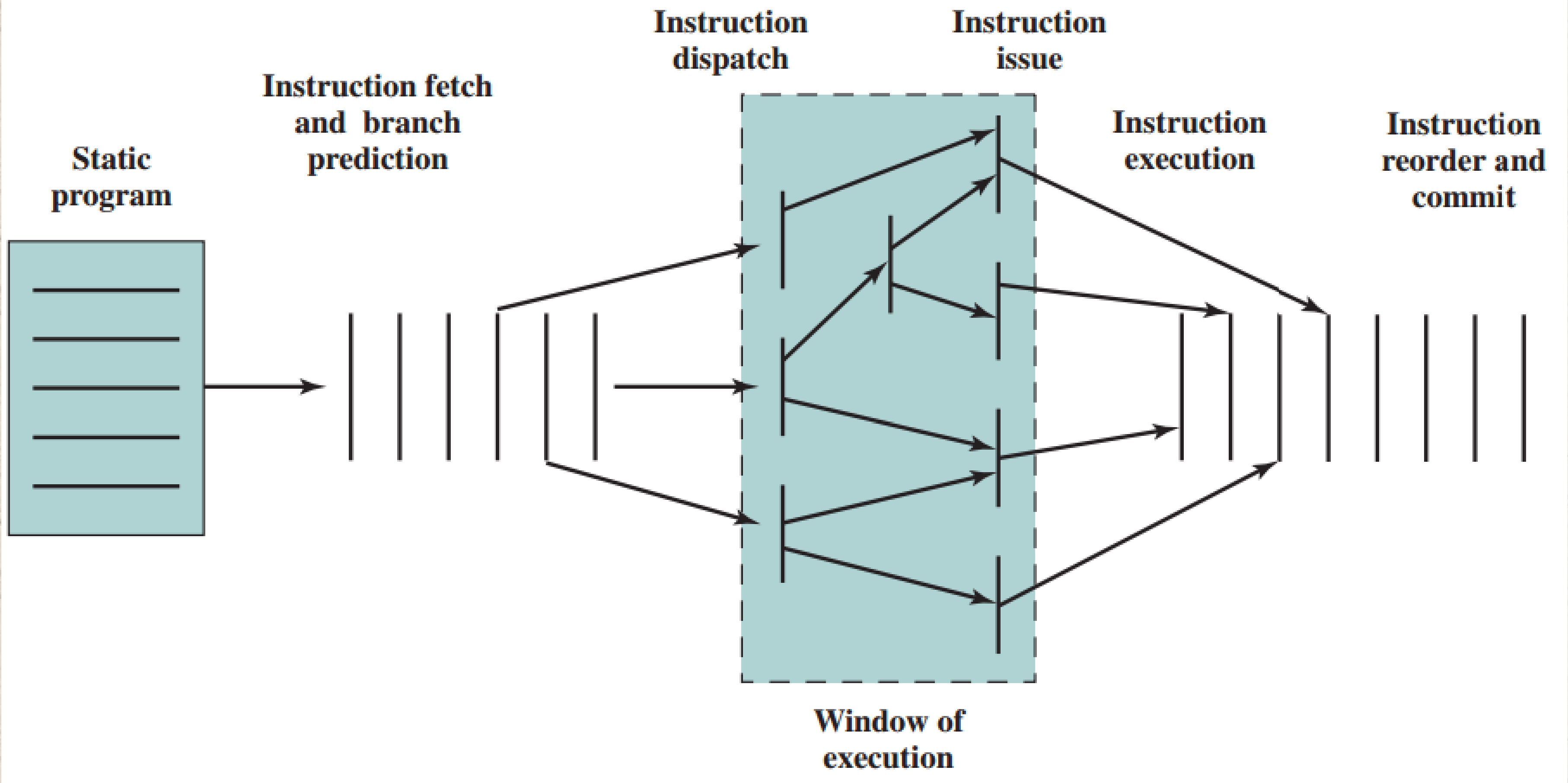


Figure 16.7 Conceptual Depiction of Superscalar Processing

+

Superscalar Implementation

- **Key elements:**
  - Instruction fetch strategies that **simultaneously fetch multiple instruction**
  - Logic for determining true dependencies involving **register values**, and **mechanisms** for communicating these **values to where they are needed** during execution
  - **Mechanisms for initiating, or issuing**, multiple instructions in parallel
  - **Resources** for parallel execution of multiple instructions, including **multiple pipelined** functional units and **memory hierarchies** capable of simultaneously servicing multiple memory references
  - **Mechanisms for committing the process state in correct order**

+

Exercises

16.1 What is the essential characteristic of the superscalar approach to processor design?
16.2 What is the difference between the superscalar and super pipelined approaches?
16.3 What is instruction-level parallelism?
16.4 Briefly define the following terms: • True data dependency • Procedural dependency • Resource conflicts • Output dependency • Antidependency
16.5 What is the distinction between instruction-level parallelism and machine parallelism?
16.6 List and briefly define three types of superscalar instruction issue policies.
16.7 What is the purpose of an instruction window?
16.8 What is register renaming and what is its purpose?
16.9 What are the key elements of a superscalar processor organization?

+

Summary

Chapter 16

Instruction-Level Parallelism and Superscalar Processors

- Superscalar versus Superpipelined

- Design issues
  - Instruction-level parallelism
  - Machine parallelism
  - Instruction issue policy
  - Register renaming
  - Branch prediction
  - Superscalar execution
  - Superscalar implementation