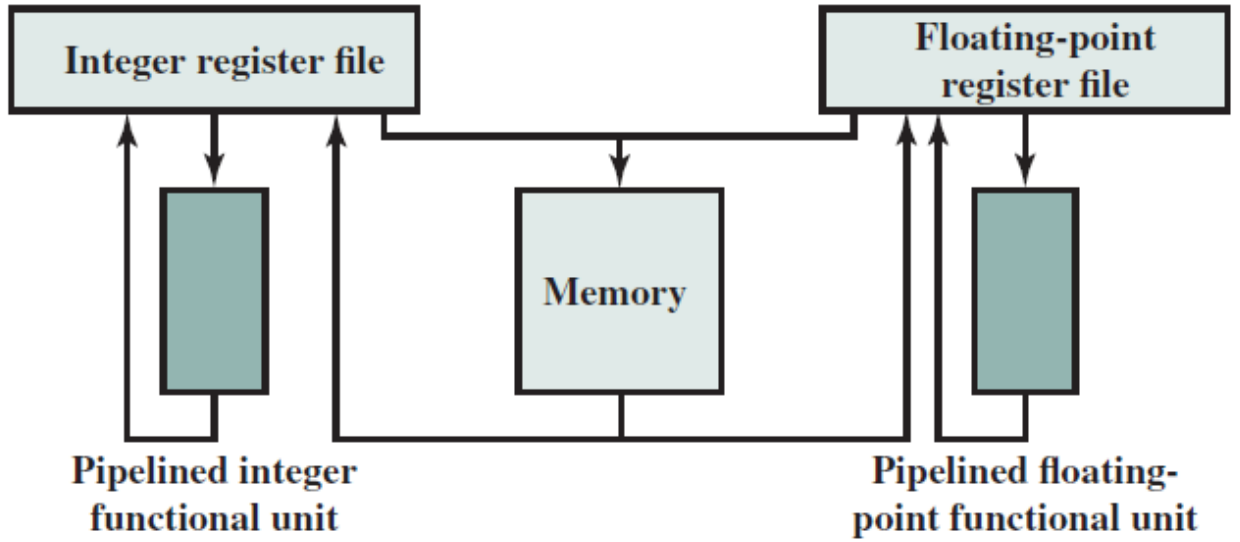
INSTRUCTION-LEVEL PARALLELISM & SUPERSCALAR PROCESSORS

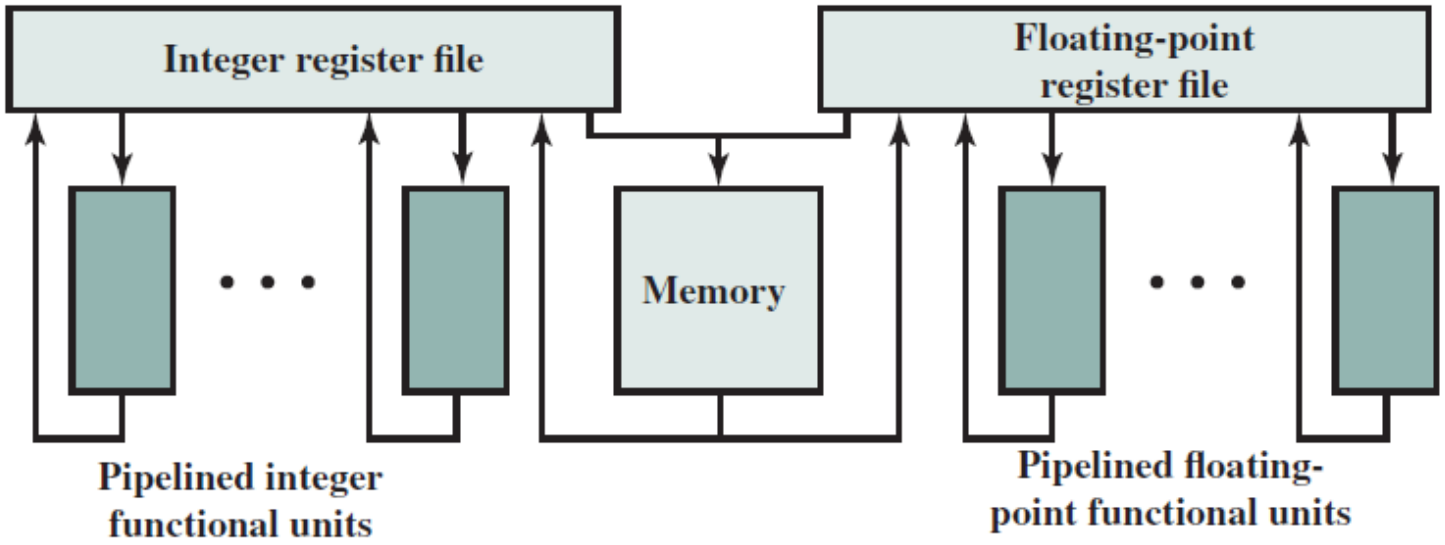
INTRODUCTION

- The term superscalar, first coined in 1987 [AGER87], refers to a machine that is designed to improve the performance of the execution of scalar instructions.
- A superscalar implementation of a processor architecture is one in which common instructions—integer and floating-point arithmetic, loads, stores, and conditional branches—can be initiated simultaneously and executed independently.
- The simplified instruction set architecture of a RISC machine lends itself readily to superscalar techniques, the superscalar approach can be used on either a RISC or CISC architecture.

INTRODUC



(a) Scalar organization

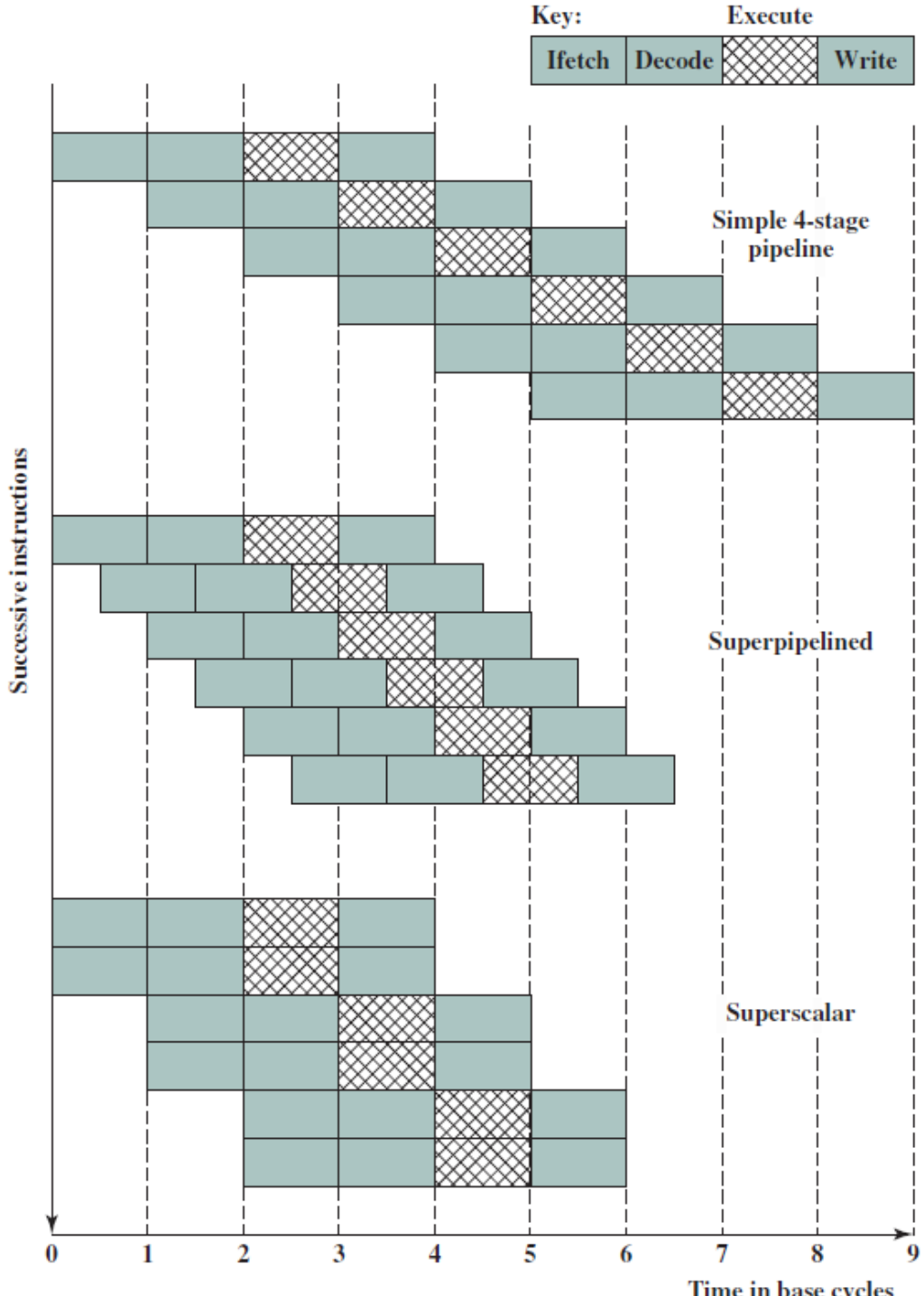


(b) Superscalar organization

Superscalar versus Superpipelined

- Superpipelining exploits the fact that many pipeline stages perform tasks that require less than half a clock cycle. Thus, a doubled internal clock speed allows the performance of two tasks in one external clock cycle.

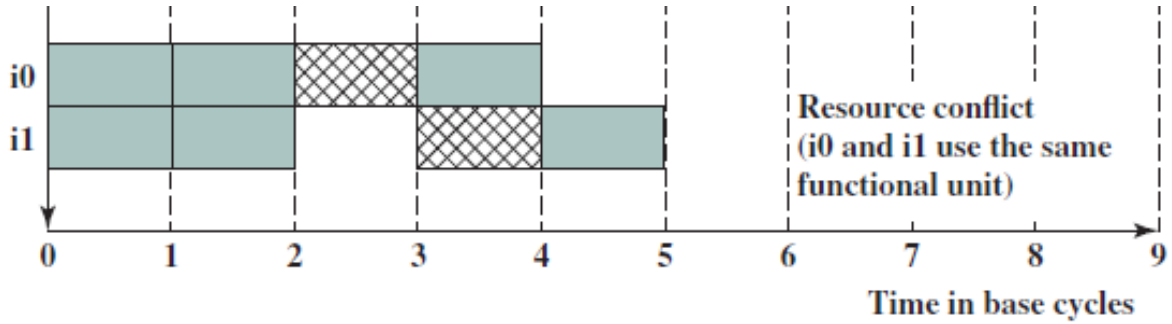
Superscalar versus Superpip

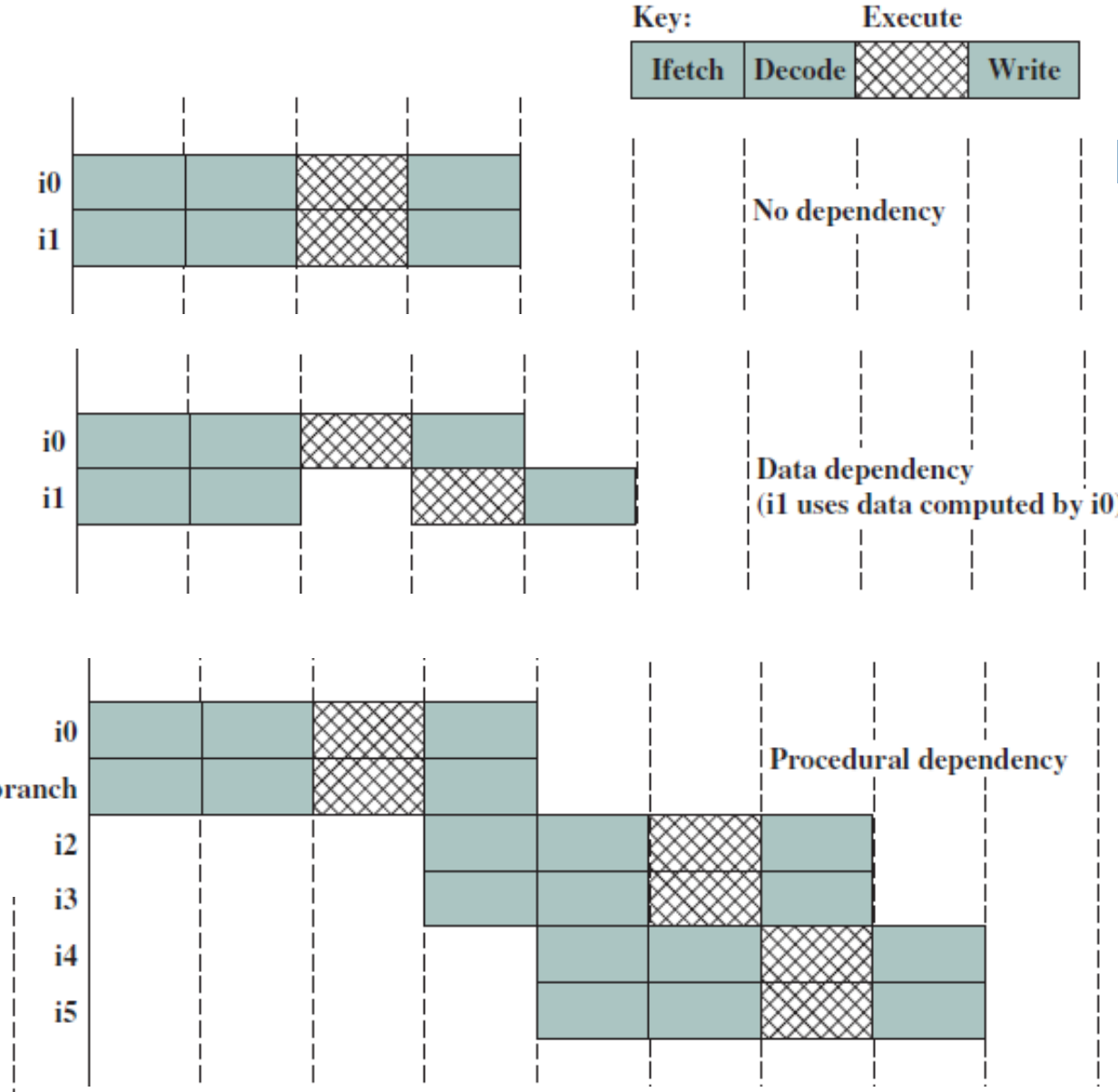


Constraints

The fundamental limitations to par

- True data dependency;
- Procedural dependency;
- Resource conflicts;
- Output dependency;
- Antidependency.





Design Issues - Instruction-Level Parallelism and Machine Parallelism

- Instruction-level parallelism exists when instructions in a sequence are independent and thus can be executed in parallel by overlapping.

```
Load R1 ← R2          Add R3 ← R3, "1"
Add R3 ← R3, "1"      Add R4 ← R3, R2
Add R4 ← R4, R2       Store [R4] ← R0
```

- Machine parallelism is a measure of the ability of the processor to take advantage of instruction-level parallelism.
- Machine parallelism is determined by the number of instructions that can be fetched and executed at the same time (the number of parallel pipelines) and by the speed and sophistication of the mechanisms that the processor uses to find independent instructions.

Instruction Issue Policy

- The term instruction issue to refer to the process of initiating instruction execution in the processor's functional units and the term instruction issue policy to refer to the protocol used to issue instructions.
- The processor is trying to look ahead of the current point of execution to locate instructions that can be brought into the pipeline and executed.
- We can group superscalar instruction issue policies 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.

Instruction Issue Policy

IN-ORDER ISSUE WITH IN-ORDER COMPLETION

- The simplest instruction issue policy is to issue instructions in the exact order that would be achieved by sequential execution (in-order issue) and to write results in that same order (in-order completion).
- In example, we assume a superscalar pipeline capable of **fetching and decoding two instructions at a time**, having **three separate functional units** (e.g., two integer arithmetic and one floating-point arithmetic), and having **two instances of the write-back pipeline stage**. The example assumes the following constraints on a six-instruction code fragment:
  - I1 requires two cycles to execute.
  - I3 and I4 conflict for the same functional unit.
  - I5 depends on the value produced by I4.
  - I5 and I6 conflict for the same functional unit.

Instruction Issue Policy

- IN-ORDER ISS
- The simplest i would be ach that same ord
- In example, w **two instructi** integer arithm **the write-bac** a six-instructio
  - I1 requires t
  - I3 and I4 co
  - I5 depends
  - I5 and I6 co

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

Instruction Issue Policy

IN-ORDER ISSUE WITH OUT-OF-ORDER COMPLETION

- Out-of-order completion is used in scalar RISC processors to improve the performance of instructions that require multiple cycles.
- Instruction I2 is allowed to run to completion prior to I1. This allows I3 to be completed earlier, with the net result of a savings of one cycle.
- With out-of-order completion, any number of instructions may be in the execution stage at any one time, up to the maximum degree of machine parallelism across all functional units.
- Instruction issuing is stalled by a resource conflict, a data dependency, or a procedural dependency.

Instruction Issue Policy

- In
  co
  pe
- In
  to
- W
  th
  m
- In
  or

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

Instruction Issue Policy

Out-of-order Issue With Out-of-order Completion

- With in-order issue, the processor will only decode instructions up to the point of a dependency or conflict. No additional instructions are decoded until the conflict is resolved.
- As a result, the processor cannot look ahead of the point of conflict to subsequent instructions that may be independent of those already in the pipeline and that may be usefully introduced into the pipeline.
- To allow out-of-order issue, it is necessary to decouple the decode and execute stages of the pipeline. This is done with a **buffer** referred to as an instruction window.
- With this organization, after a processor has finished decoding an instruction, it is placed in the instruction window. As long as this buffer is not full, the processor can continue to fetch and decode new instructions.
- When a functional unit becomes available in the execute stage, an instruction from the instruction window may be issued to the execute stage.

Instruction Issue Policy

- Out-of-order Issue With Out-of-order Completion: With in-order issue, the p
- A
- T
- V

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

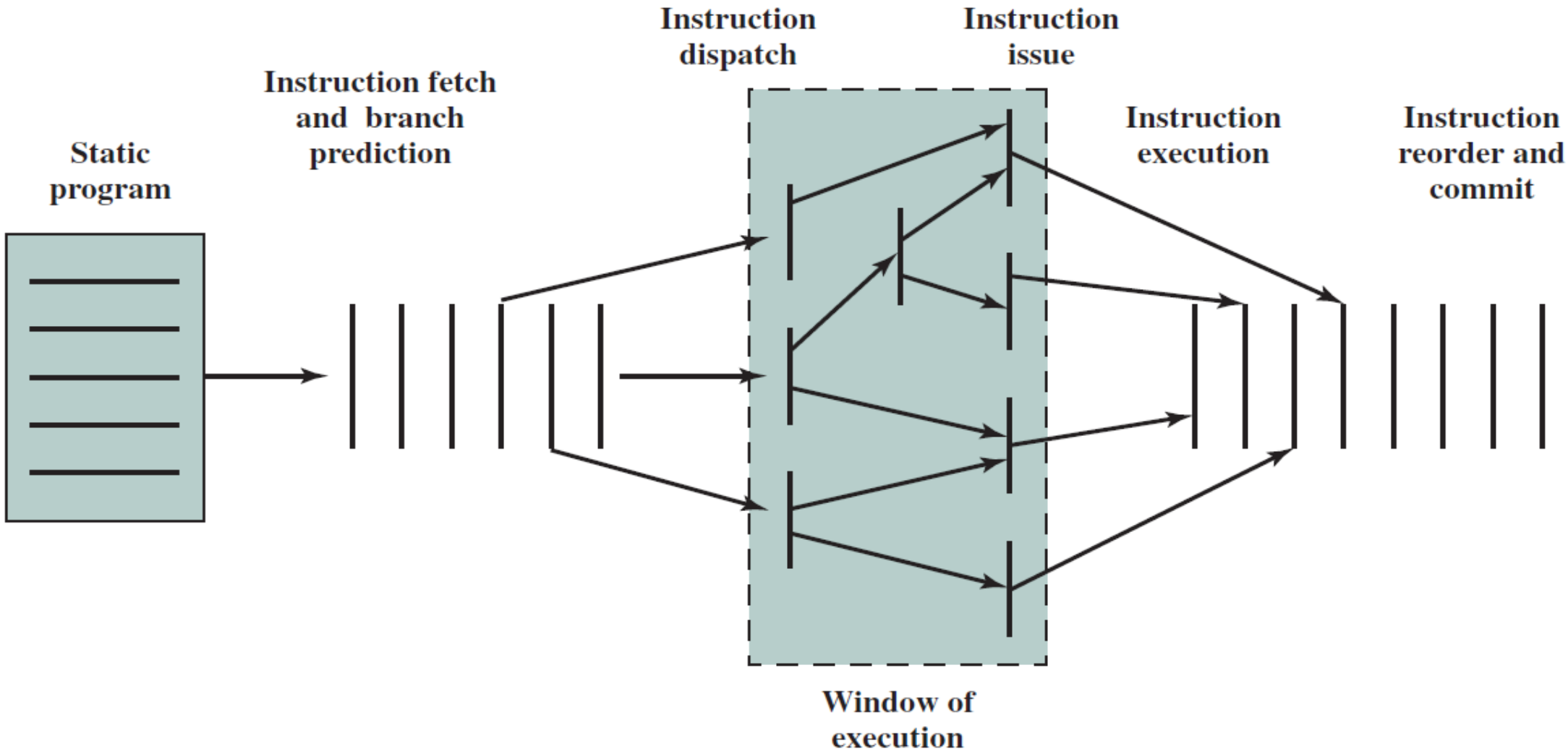
(c) Out-of-order issue and out-of-order completion

- When a functional unit becomes available in the execute stage, an instruction from the instruction window may be issued to the execute stage.

Branch Prediction

- Any high-performance pipelined machine must address the issue of dealing with branches. The Intel 80486 addressed the problem by fetching both the next sequential instruction after a branch and speculatively fetching the branch target instruction. However, because there are two pipeline stages between prefetch and execution, this strategy incurs a two-cycle delay when the branch gets taken.
- With the advent of RISC machines, the delayed branch strategy was explored. This allows the processor to calculate the result of conditional branch instructions before any unusable instructions have been prefetched. With this method, the processor always executes the single instruction that immediately follows the branch.

Superscalar Execution



Superscalar Implementation

Key elements of the processor hardware required for the superscalar approach are

- Instruction fetch strategies that simultaneously fetch multiple instructions, often by predicting the outcomes of, and fetching beyond, conditional branch instructions. These functions require the use of multiple pipeline fetch and decode stages, and branch prediction logic.
- 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.