Computer Architecture

Instruction-level Parallelism And Superscalar Processors

Md. Biplob Hosen
Lecturer, IIT-JU

Reference Books

- Computer Organization and Architecture: Designing for Performance- William Stallings (8^{th} Edition)
  - Any later edition is fine

Superscalar Processor

- In which multiple independent instruction pipelines are used.
- Each pipeline consists of several stages, so that each pipeline can handle multiple instructions at a time.
- Multiple pipeline introduce a new level of parallelism, enabling multiple streams of instructions to be processed at a time.
- A superscalar processor exploits which is known as **instruction-level parallelism**, which refers to the degree to which the instructions of a program can be executed in parallel.

Superscalar Processor

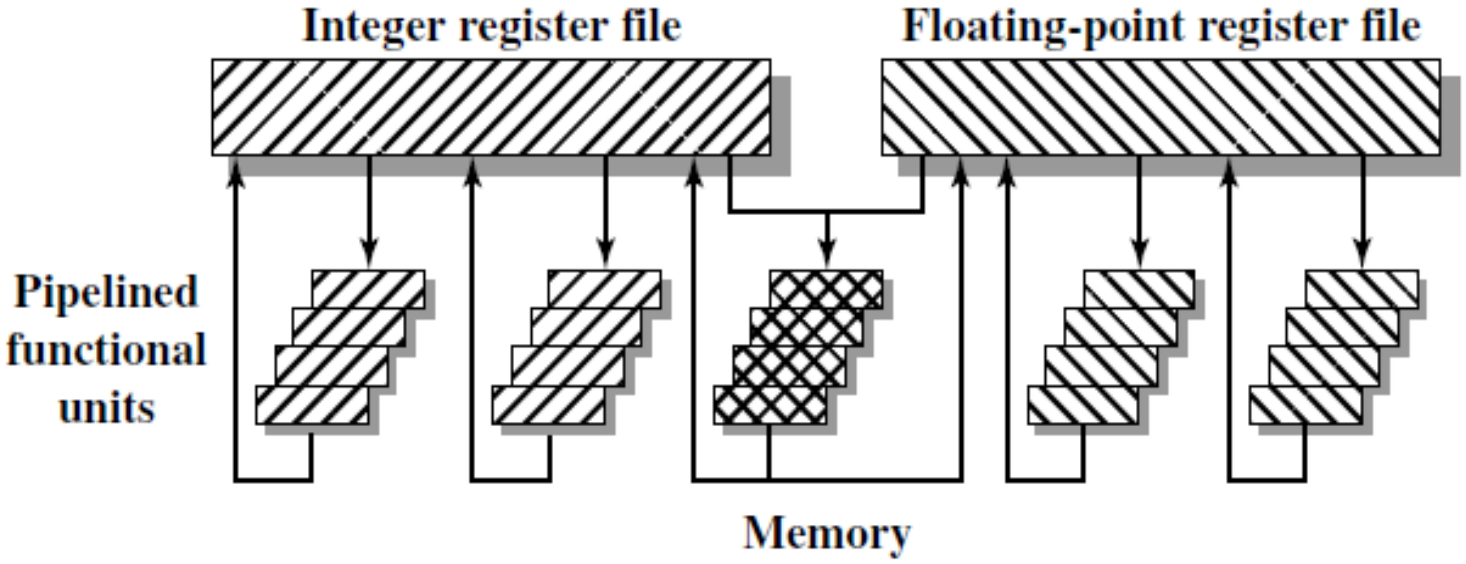
- A superscalar processor typically fetches multiple instructions at a time and then attempts to find nearby instructions that are independent of one-another, and can therefore be executed in parallel.
- if the input to one instruction depends on the output of preceding instruction, then the later instruction can not execute at the same time or before the former instruction.
- once such dependencies have been identified, the processor may issue and complete instructions in an order that differs from that of the original machine code.
- The processor may eliminate some unnecessary dependencies by the use of additional registers and the renaming of register references in the original code.
- Whereas pure RISC processors often employ delayed branches to maximize the utilization of instruction pipeline, this method is less appropriate in superscalar processor.
- Instead, most superscalar machines use traditional branch prediction methods to improve efficiency.

Superscalar Architecture

- A processor architecture
- In which common instructions—integer and floating-point arithmetic, loads, stores, and conditional branches—can be initiated simultaneously and executed independently.
- Refers to a machine that is designed to improve the performance of the execution of scalar instructions.
  - In most applications, the bulk of the operations are on scalar quantities.

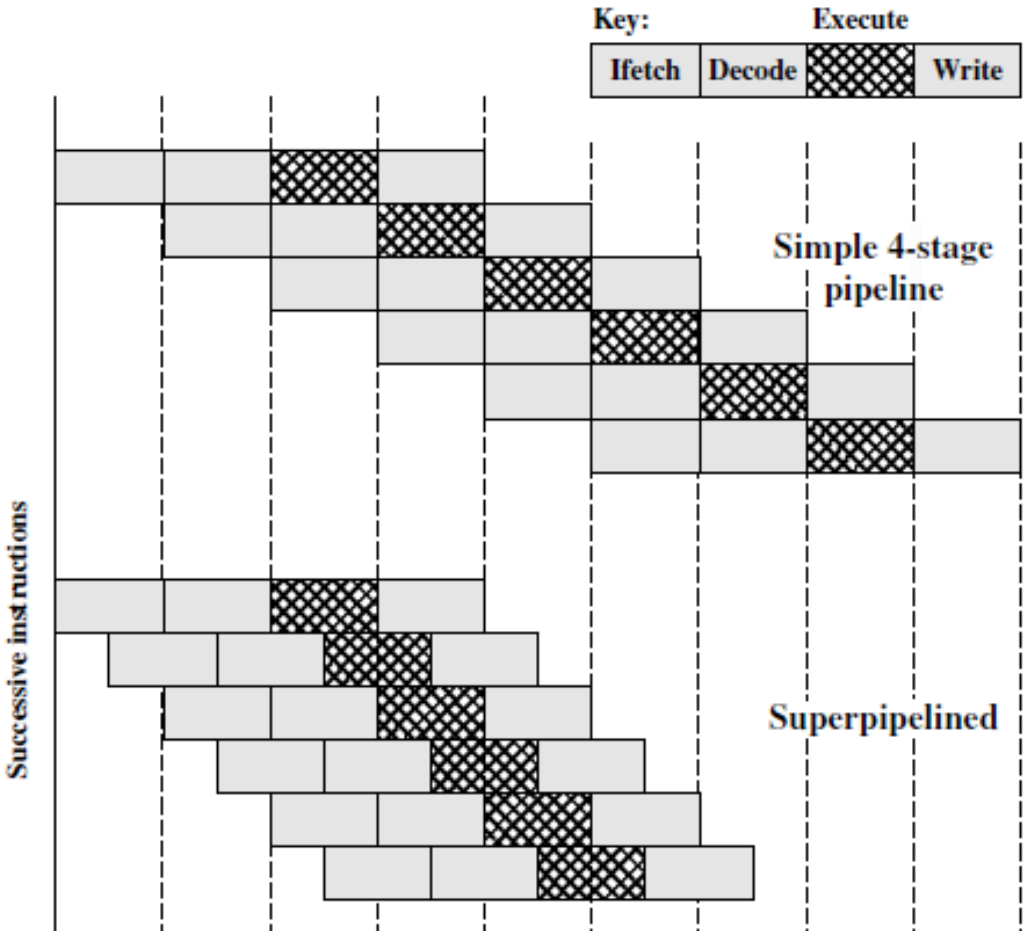
Superscalar Architecture

- Following figure shows, in general terms, the superscalar approach.
- There are multiple functional units, each of which is implemented as a pipeline, which supports parallel execution of several instructions.
- In this example, two integer, two floating-point, and one memory (load/store) operations can be executed at the same time.

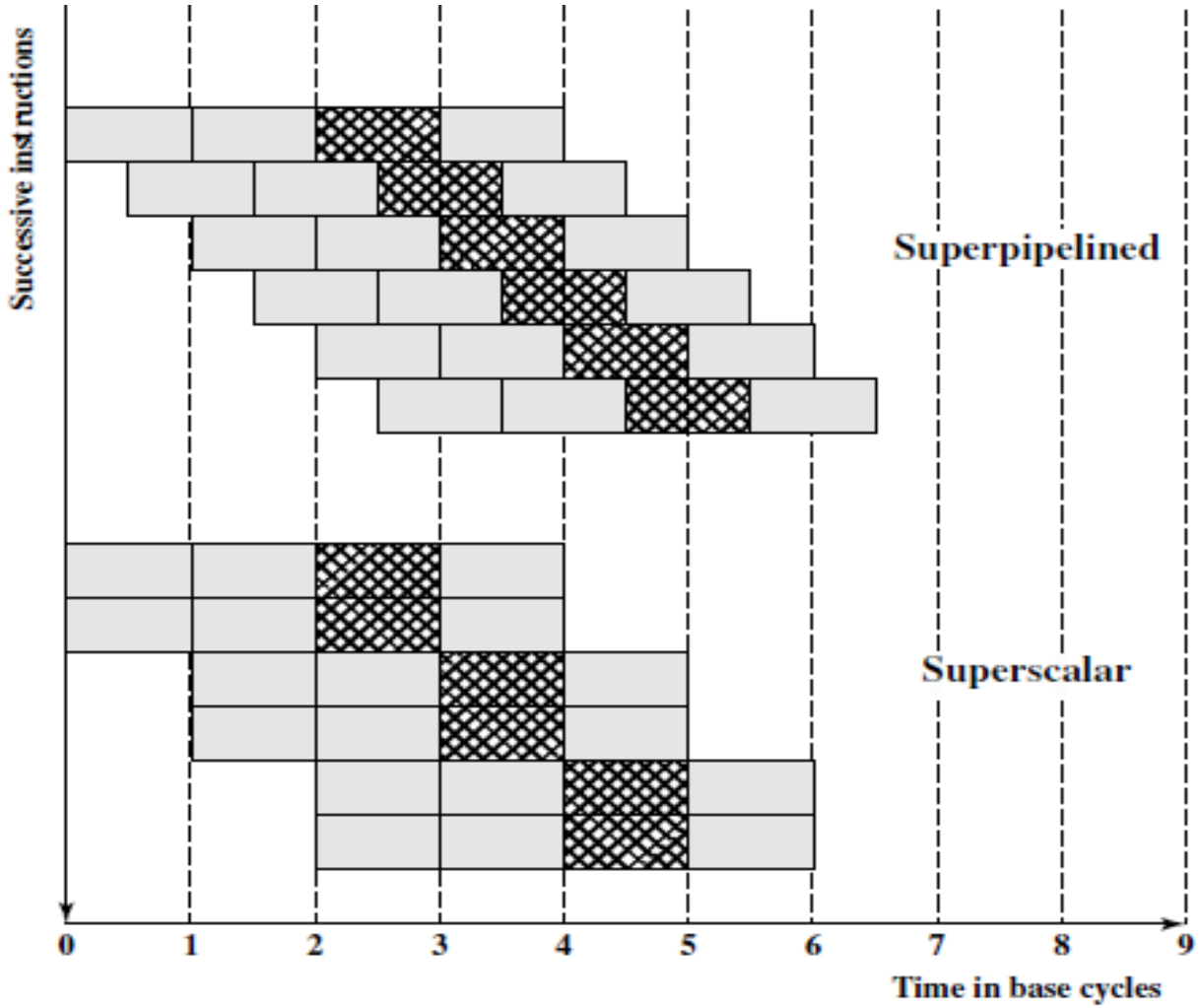


Superpipelined Architecture

- 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 vs. Superpipelined



Limitations

- The superscalar approach depends on the ability to execute multiple instructions in parallel.
- The term instruction-level parallelism refers to the degree to which, on average, 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.
- Obstacles for achieving parallelism:
  - True data dependency
  - Procedural dependency
  - Resource conflicts
  - Output dependency
  - Anti-dependency

True Data Dependency

- Consider the following sequence:
  - ADD EAX, ECX (I0)
  - MOV EBX, EAX (I1)
- The second instruction can not be executed until the first instruction execute.
- The reason is second instruction needs data produced by the first instruction.
- This situation is referred as true data dependency (also called flow dependency or write after read).
- Figure illustrates this dependency in a superscalar machine of degree 2.
- With no dependency, two instructions can be fetched and executed parallel.
- If there is a data dependency between between the instructions, tjem the second instruction os delayed as many clock cycles as required to remove this dependencies.
- In general, any instruction must be delayed until all of its input values have been produced.

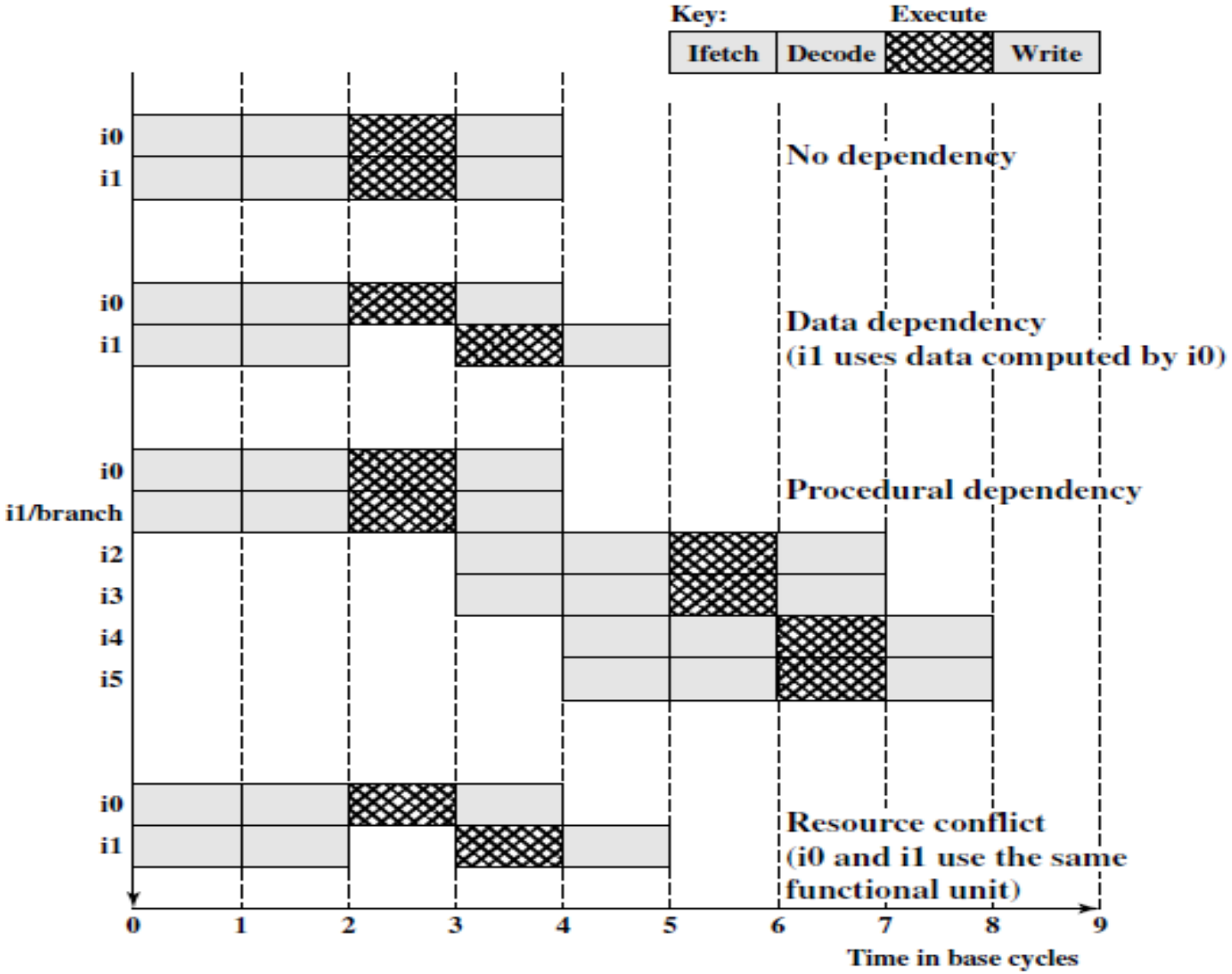
Procedural Dependency

- The presence of branches in an instruction sequence complicates the pipeline operation.
- The instruction following a branch (taken or not taken) have a procedural dependency on the branch and can not be executed until the branch is executed.
- Following figure illustrates the effect of a branch on a superscalar pipeline of degree-2.
- As we have seen, this type of procedural also affects a scalar pipeline.
- The consequence for a superscalar pipeline is more severe, because a greater magnitude of opportunity is lost each delay.
- If variable-length instructions are used, then another sort of procedural dependency arises.
- Because the length of any particular instruction is not known, it must be at least partially decoded before the following instruction can be fetched.
- This prevents the simultaneous fetching required in a superscalar pipeline.
- This is one of the reasons that superscalar techniques are more applicable to a RISC architecture, with its fixed instruction length.

Resource Dependency

- A resource conflict is a competition of two or more instructions for the same resource at the same time.
- Examples of resources include memories, caches, buses, register-file ports functional units (e.g. ALU adder).
- In terms of the pipeline, a resource conflict exhibits similar behavior to a data dependency.
- There are some difference however.
- For one thing, resource conflicts can be overcome by duplication of resources, whereas a true data dependency can not be eliminated.
- Also, when an operation takes a long time to complete, resource conflicts can be minimized by pipelining the appropriate functional unit.

Effect of Dependencies



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.
  - Consider the following two code fragments:

Load R1 ← R2 Add R3 ← R3, “1” Add R4 ← R4, R2	Add R3 ← R3, “1” Add R4 ← R3, R2 Store [R4] ← R0

  - The three instructions on the left side are independent, and all three can be executed parallel.
  - In contrast, all three instructions on the right can not be executed in parallel, because the 2nd instruction uses the result of the 1st, and the 3rd one uses the result of the 2nd.

Design Issues

- The degree of instruction-level parallelism is determined by the frequency of **true data dependencies** and **procedural dependencies** in the code.
- These factors, in turn, are dependent on the instruction set architecture and on the application.
- Also determined by **operation latency**: the time until the result of an instruction is available for use as an operand in a subsequent instruction.
- The latency determines how much of a delay a data or procedural latency will cause.

Design Issues

- Machine parallelism is a measure of the ability of the processor to take advantage of instruction-level parallelism.
  - 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.

Both instruction level parallelism and machine level parallelism are important for enhancing performance.

Design Issues

- Instruction Issue Policy: The processor must be able to identify instruction level parallelism and co-ordinate the fetching, decoding, and execution of instructions in parallel.
- 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.
- In general, we can say that instruction issue occurs when instruction moves from the decode stage of the pipeline to the first execute stage of the pipeline.

Design Issues

- The processor always try to look ahead of the current point of execution to locate instructions that can be brought into the pipeline and executed.
- The types of ordering are important for this regard:
- The order in which instructions are fetched.
- The order in which instructions are executed.
- The order in which instructions update the contents of registers and memory locations.

Design Issues

- Superscalar instruction issue policies fall 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

Design Issues

In-order issue with in-order completion

- Consider the following constraints of on a six-instruction code fragment where we have three functional units:
  - I1 requires two cycles to execute.
  - I3, I4 conflict for same functional unit
  - I5 depends on the value produced by I4.
  - I5, I6 conflict for same functional unit

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

Design Issues

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

Design Issues

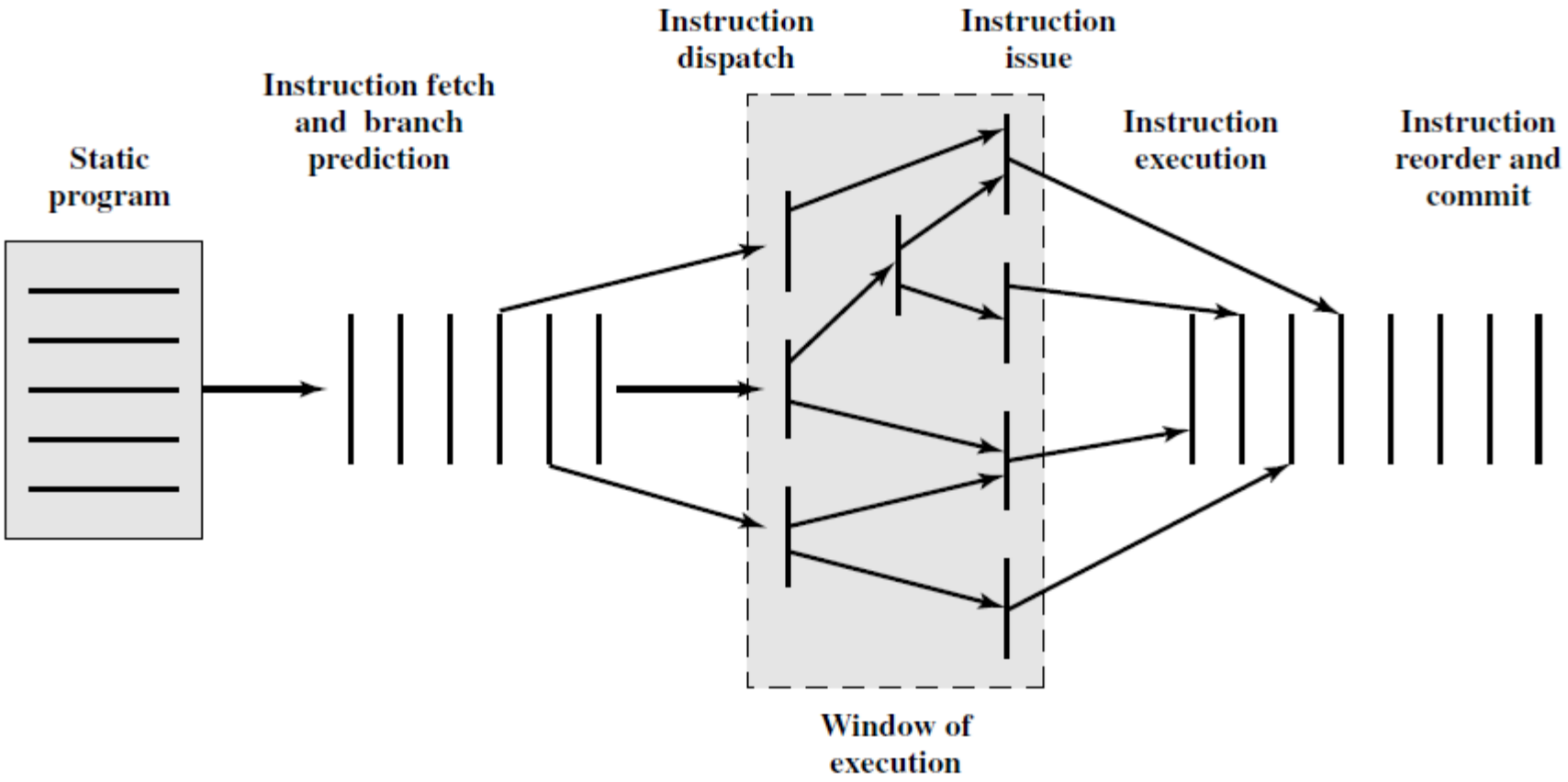
- Register Renaming: When out-of-order techniques are used, the values in registers cannot be fully known at each point in time just from a consideration of the sequence of instructions dictated by the program.
- Solution: Duplication of resources.
  - Also refers to as register renaming.
- When a new register value is created, a new register is allocated for that value.

Design Issues

- Subsequent instructions that access that value as a source operand in that register must go through a renaming process:
  - The register references in those instructions must be revised to refer to the register containing the needed value .
  - The same original register reference in several different instructions may refer to different actual registers, if different values are intended.

Design Issues

- Superscalar Execution:



Design Issues

- Superscalar Implementation Issues:

a) 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.

b) Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution.

c) Mechanisms for initiating, or issuing, multiple instructions in parallel.

d) Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references.

e) Mechanisms for committing the process state in correct order.

Practice Problems

- Problems: 14.1, 14.3, 14.4

Thank you!