电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC4019: Real-Time Computer Systems and Architecture

Lecture 17

Instruction-Level Parallelism, and Superscalar Processors (Part-1)

Superscalar Overview

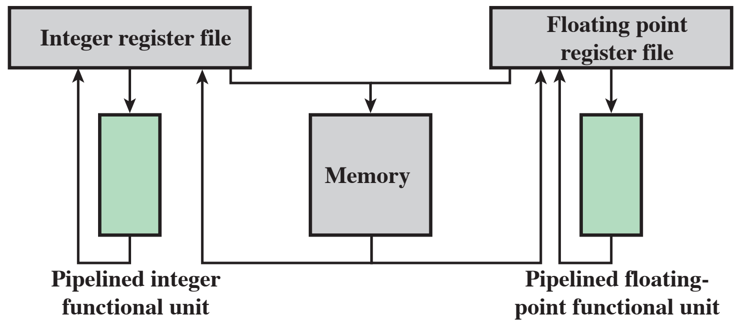
- Term first coined in 1987
- Superscalar refers to a machine that is designed to improve the performance of the execution of scalar instructions
- In most applications the bulk of the operations are on scalar quantities
- Represents the next step in the evolution of high-performance general-purpose processors
- Essence of the approach is the ability to execute instructions independently and concurrently in different pipelines
- Concept can be further exploited by allowing instructions to be executed in an order different from the program order

Superscalar Organization Compared to Ordinary Scalar Organization

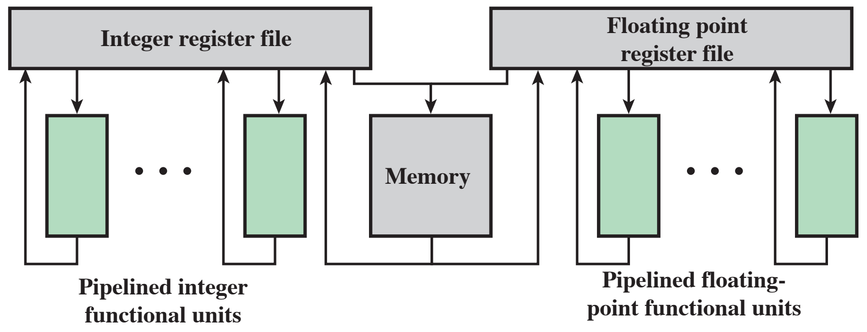
In a traditional scalar organization, there is a single pipelined functional unit for integer operations and one for floating-point operations

Parallelism is achieved by enabling multiple instructions to be at different stages of the pipeline at one time

In the superscalar organization, there are multiple functional units, each of which is implemented as a pipeline


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

Superscalar vs Ordinary Scalar

- Each individual functional unit provides a degree of parallelism by virtue of its pipelined structure.
- The use of multiple functional units enables the processor to execute streams of instructions in parallel, one stream for each pipeline.
- It is the responsibility of the hardware, in conjunction with the compiler, to assure that the parallel execution does not violate the intent of the program.

Superpipelining

- An alternative approach to achieving greater performance is referred to as super-pipelining, a term first coined in 1988
- 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
- We have seen one example of this approach with the MIPS R4000

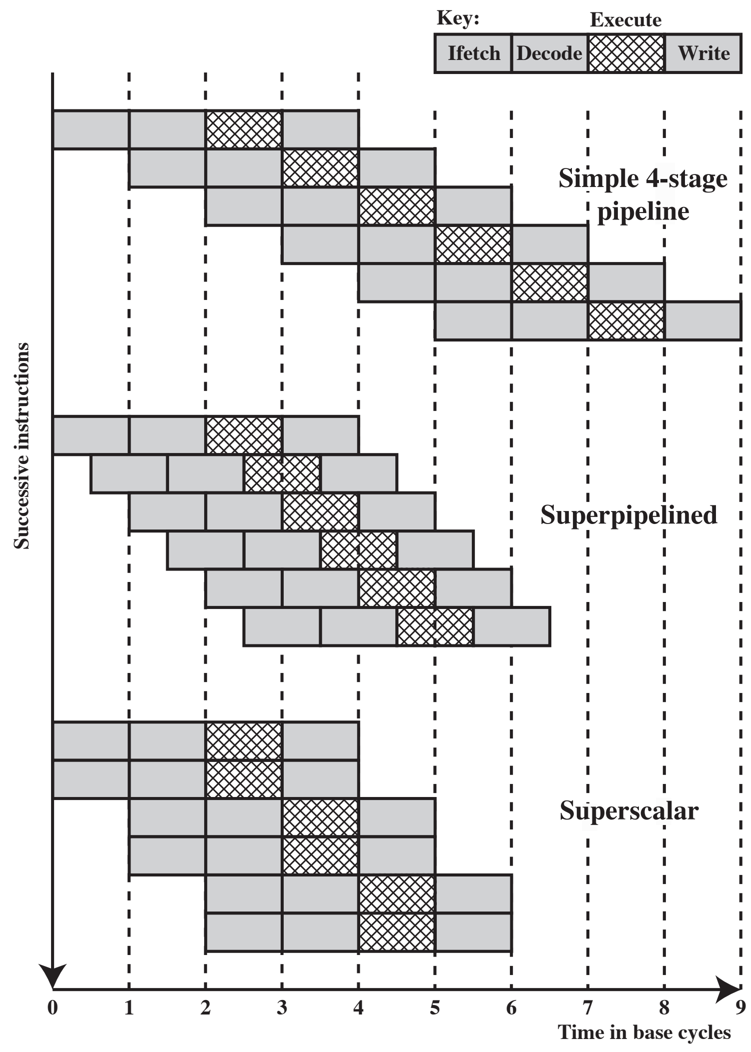
Comparison of Superscalar and Super Pipeline Approaches (1 of 3)

The upper part of the diagram illustrates an ordinary pipeline, used as a base for comparison

The base pipeline issues one instruction per clock cycle and can perform one pipeline stage per clock cycle

The pipeline has four stages: instruction fetch, operation decode, operation execution, and result write back. The execution stage is crosshatched for clarity.

Note that although several instructions are executing concurrently, only one instruction is in its execution stage at any one time.

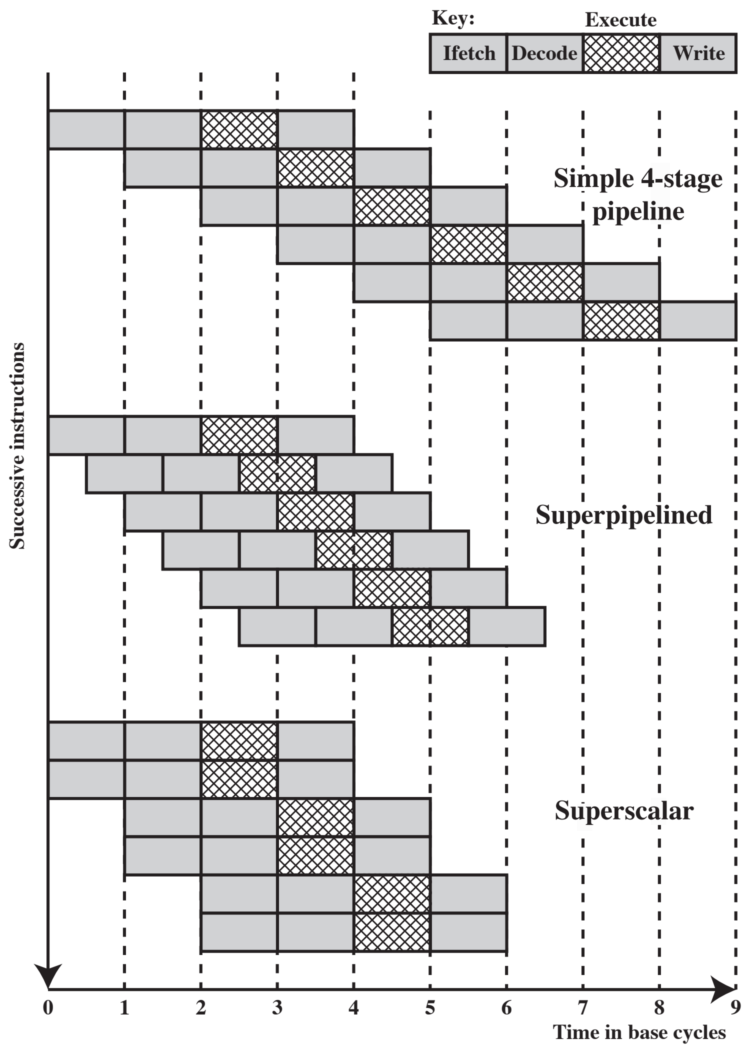


Comparison of Superscalar and Super Pipeline Approaches (2 of 3)

The next part of the diagram shows a superpipelined implementation that is capable of performing two pipeline stages per clock cycle.

An alternative way of looking at this is that the functions performed in each stage can be split into two non-overlapping parts and each can execute in half a clock cycle.

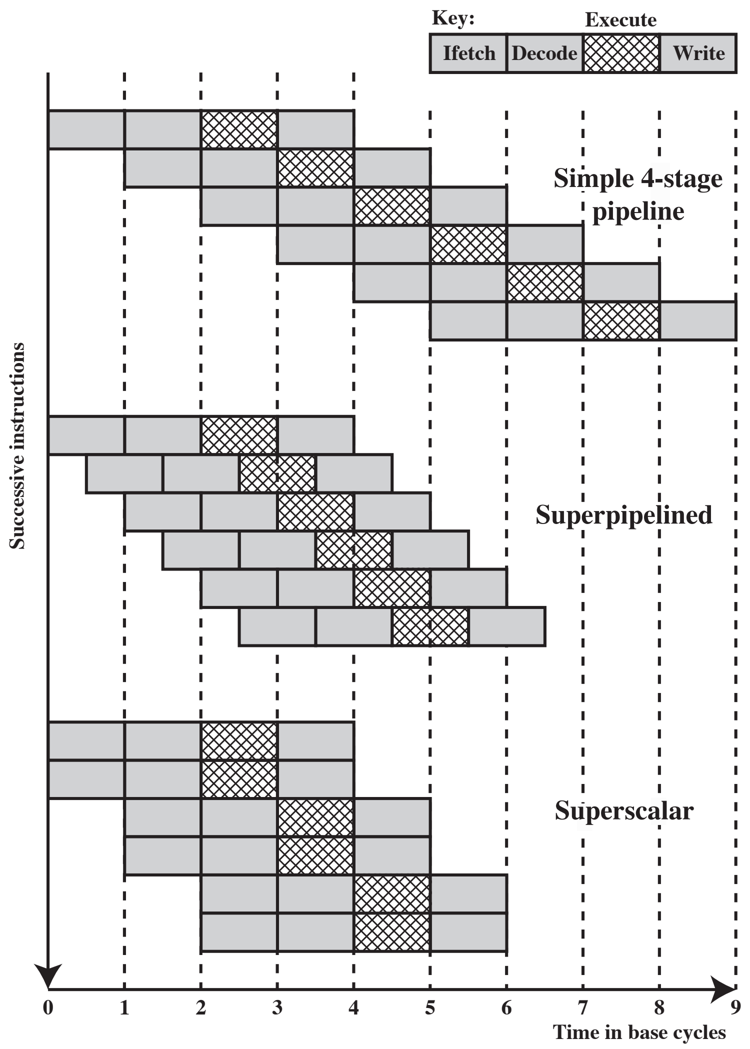
A superpipeline implementation that behaves in this fashion is said to be of degree 2.



Comparison of Superscalar and Super Pipeline Approaches (3 of 3)

Finally, the lowest part of the diagram shows a superscalar implementation capable of executing two instances of each stage in parallel.

Higher-degree superpipeline and super-scalar implementations are of course possible.



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
  - Antidependency

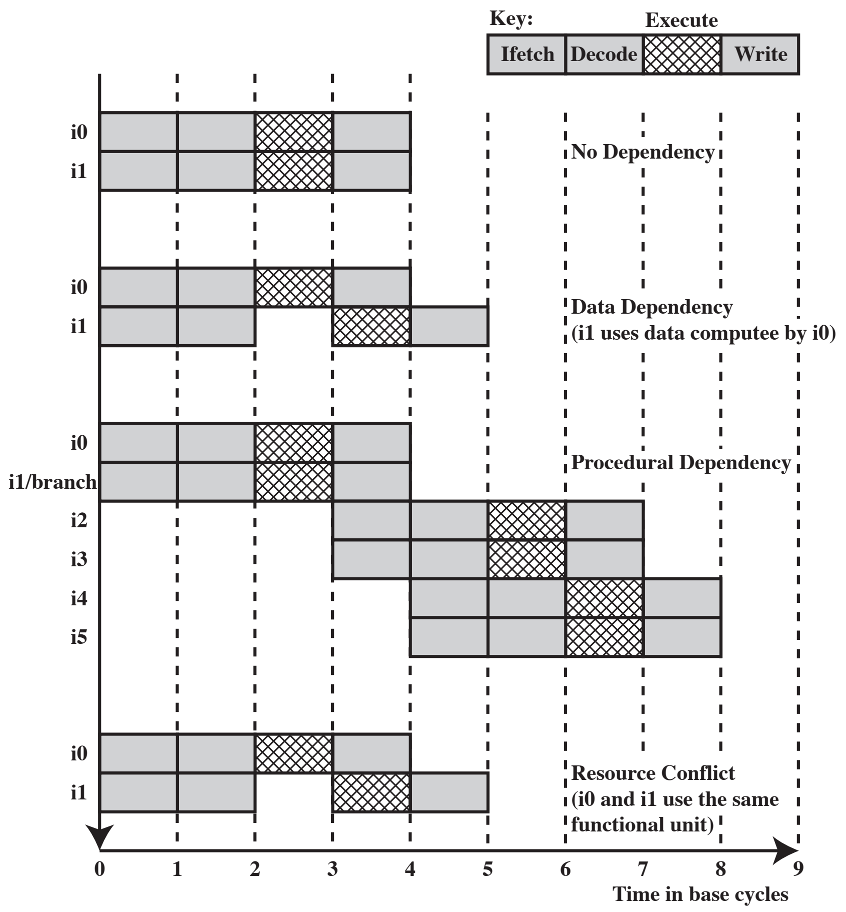
Effect of Dependencies

The presence of branches in an instruction sequence complicates the pipeline operation

The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot be executed until the branch is executed

Figure illustrates the effect of a branch on a superscalar pipeline of degree 2

The consequence of procedural dependency for a superscalar pipeline is more severe, because a greater magnitude of opportunity is lost with each delay



Design Issues

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

Instruction Issue Policy (1 of 2)

- 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

Instruction Issue Policy (2 of 2)

- 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

Figure gives an example of this policy.

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:

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

Register Renaming

- Output and anti dependencies occur because register contents may not reflect the correct ordering from the program
- May result in a pipeline stall
- Registers allocated dynamically

Branch Prediction (1 of 2)

- 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

Branch Prediction (2 of 2)

  - Keeps the pipeline full while the processor fetches a new instruction stream
- Superscalar machines:
  - Delayed branch strategy has less appeal
  - Have returned to pre-RISC techniques of branch prediction