Instruction Level Parallelism and Superscalar Processors

Contents of Lecture:

- Design Issues
  - ✓ Instruction-Level Parallelism and Machine Parallelism
  - ✓ Instruction Issue Policy

References for This Lecture:

- ✓ William Stallings, Computer Organization and Architecture Designing For Performance, 9th Edition, Chapter 16: *Instruction Level Parallelism and Superscalar Processors*

Instruction-Level Parallelism and Machine Parallelism:

Instruction-level parallelism:

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

- As an example of the concept of instruction-level parallelism, consider the following code:

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

- The three instructions on the left are independent, and in theory all three could be executed in parallel. In contrast, the three instructions on the right cannot be executed in parallel because the second instruction uses the result of the first, and the third instruction uses the result of the second.

- 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.

- Instruction-level parallelism is also determined by what refers to as 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 dependency will cause.

Machine parallelism:

- 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.

- **Both instruction-level and machine parallelism:**
  - ✓ Are important factors in enhancing performance.
  - ✓ A program may not have enough instruction-level parallelism to take full advantage of machine parallelism.
  - ✓ The use of a fixed-length instruction set architecture, as in a RISC, enhances instruction-level parallelism.
  - ✓ On the other hand, limited machine parallelism will limit performance no matter what the nature of the program.

Instruction Issue Policy:

- Machine parallelism is not simply a matter of having multiple instances of each pipeline stage. The processor must also be able to identify instruction-level parallelism and orchestrate the fetching, decoding, and execution of instructions in parallel.

- 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.
- 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.

- In essence, 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.

- Three types of orderings are important in 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 register and memory locations (order of completion)

- The more sophisticated the processor, the less it is bound by a strict relationship between these orderings.

- To optimize utilization of the various pipeline elements, the processor will need to alter one or more of these orderings with respect to the ordering to be found in a strict sequential execution.

- The one constraint on the processor is that the result must be correct. Thus, the processor must accommodate the various dependencies and conflicts discussed earlier.

- In general terms, 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

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).

- Following 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:
  - ✓ 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 a 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

In-order issue and in-order completion

- Instructions are fetched two at a time and passed to the decode unit. Because instructions are fetched in pairs, the next two instructions must wait until the pair of decode pipeline stages has cleared.
- To guarantee in-order completion, when there is a conflict for a functional unit or when a functional unit requires more than one cycle to generate a result, the issuing of instructions temporarily stalls.
  - ✓ In this example, the elapsed time from decoding the first instruction to writing the last results is eight cycles.

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.

- Following figure illustrates its use on a superscalar processor.
  - ✓ 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.

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

In-order issue and out-of-order completion

- 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.
- In addition to the aforementioned limitations, a new dependency, which we referred to earlier as an output dependency (also called write after write [WAW] dependency), arises.

- The following code fragment illustrates this dependency (**op** represents any operation):

  I1: R3 ←R3 **op** R5
  I2: R4 ←R3 + 1
  I3: R3 ←R5 + 1
  I4: R7 ←R3 **op** R4

- Instruction I2 cannot execute before instruction I1, because it needs the result in register R3 produced in I1; this is an example of a true data dependency. Similarly, I4 must wait for I3, because it uses a result produced by I3.

- What about the relationship between I1 and I3?
  - ✓ There is no data dependency here, as we have defined it.
- However, if I3 executes to completion prior to I1, then the wrong value of the contents of R3 will be fetched for the execution of I4. Consequently, I3 must complete after I1 to produce the correct output values. To ensure this, the issuing of the third instruction must be stalled if its result might later be overwritten by an older instruction that takes longer to complete.

- Out-of-order completion requires more complex instruction issue logic than in-order completion. In addition, it is more difficult to deal with instruction interrupts and exceptions. When an interrupt occurs, instruction execution at the current point is suspended, to be resumed later.

- The processor must assure that the resumption takes into account that, at the time of interruption, instructions ahead of the instruction that caused the interrupt may already have completed.

Out-of-order issue with out-of-order completion:

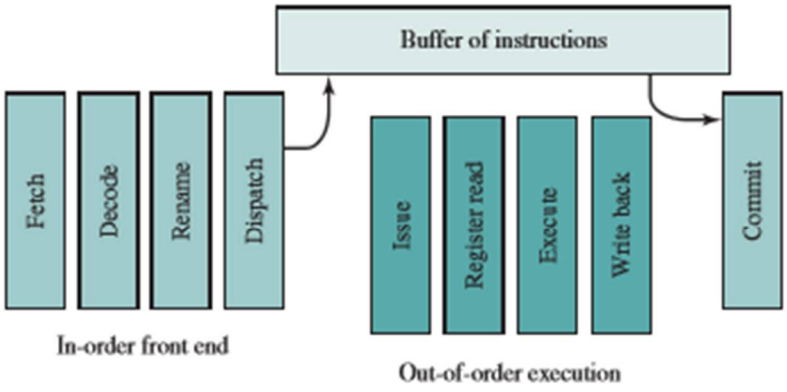
- With in-order issue, the processor will only decode instructions up to the point of a dependency or conflict. No additional instructions are decoded until the conflict is resolved. As a result, the processor cannot look ahead of the point of conflict to subsequent instructions that may be independent 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.

- Following figure suggests this organization. The result of this organization is that:
  - ✓ The processor has a lookahead capability, allowing it to identify independent instructions that can be brought into the execute stage.
  - ✓ Instructions are issued from the instruction window with little regard for their original program order.
  - ✓ As before, the only constraint is that the program execution behaves correctly.



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

❖ Following figures illustrates this policy:

- ✓ During each of the first three cycles, two instructions are fetched into the decode stage.
- ✓ During each cycle, subject to the constraint of the buffer size, two instructions move from the decode stage to the instruction window.
- ✓ In this example, it is possible to issue instruction I6 ahead of I5 (recall that I5 depends on I4, but I6 does not). Thus, one cycle is saved in both the execute and write-back stages, and the end-to-end savings, compared with Figure 16.4b, is one cycle
- ✓ The instruction window is depicted to illustrate its role. However, this window is not an additional pipeline stage. An instruction being in the window simply implies that the processor has sufficient information about that instruction to decide when it can be issued.

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

Out-of-order issue and out-of-order completion