

Superscalar (III)

Instruction Issue Policy (cont.)

- Example:
 - Assume a superscalar pipeline is capable of fetching and decoding 2 instructions at a time
 - Instructions are fetched and decoded in pair. The next two instructions must wait to be decoded until the pair of decode pipeline stages has cleared.
 - having 3 separate ALUs (e.g., two for integer arithmetic and one for floating-point arithmetic)
 - 2 instances of the write-back pipeline stage
 - 6 instruction code fragment with the following constraints:
 - I1 requires two cycles to execute
 - I3 and I4 conflict for the same ALU (e.g., both need floating-point arithmetic)
 - I5 depends on the value produced by I4
 - I5 and I6 conflict for an ALU (which may be different from the one I3 and I4 need)
 - To fetch, decode, and write back an instruction, each stage need 1 clock cycle.
 - When there is a conflict for a functional unit, or when a functional unit requires more than one cycle to generate a result, instructions temporarily stall.

In-order issue, 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.
- *I2 in the above example is allowed to run to completion prior to I1. This allows I3 to be completed earlier. Save one cycle.*

Fetch		Decode		ALU1	ALU2	ALU3	WriteBack1	WriteBack2	Cycle
I1	I2								1
I3	I4	I1	I2						2
I5	I6	I3	I4	I1	I2				3
(I5)	(I6)		(I4)	I1		I3	I2		4
		I5	I6			I4	I1	I3	5
			(I6)			I5	I4		6

Fetch		Decode		ALU1	ALU2	ALU3	WriteBack1	WriteBack2	Cycle
						I6	I5		7
							I6		8

- Note: in this example, I3 can use ALU2 or ALU3 at cycle 4. This example lets I3 use ALU3. I5 can use ALU1, ALU2 or ALU3. This example lets I5 use ALU3.

Out-of-order issue, out-of-order completion

- With **in-order issue**, the processor will only **decode** instructions **up to the point of true data dependency or resource conflict**.
- The processor **cannot look ahead beyond the point of conflict** to subsequent instructions that may be independent of those already in the pipeline that may be usefully introduced into the pipeline.
- It is necessary to **decouple the decode and execute stages**.
- This is done with a buffer, **instruction window**.
 - After a processor has finished decoding an instruction, the decoded instruction 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 can be issued** to the execute stage. Any instruction may be issued, provided that:
 - it needs the particular **functional unit** that is **available**
 - **no conflicts or dependencies** block this instruction

Fetch		Decode		Window	ALU1	ALU2	ALU3	WriteBack1	WriteBack2	Cycle
I1	I2									1
I3	I4	I1	I2							2
I5	I6	I3	I4	I1, I2	I1	I2				3
		I5	I6	I3, I4	I1		I3	I2		4
				I5, I6		I6	I4	I1	I3	5
						I5		I4	I6	6
								I5		7

- Note: in this example, I3 can use ALU2 or ALU3 at cycle 4. This example lets I3 use ALU3. I6 can use ALU1 or ALU2 at cycle 5. This example lets I6 use ALU2. I5 and I6 don't have

functional unit conflict with I4, it's unnecessary to let them use ALU3 here, as it slows down the execution for one more cycle. This example assumes that the window is large enough to buffer all decoded instructions.

Discussions:

- In-order-issue and in-order completion are easier to implement as they make it easier to define the state of the computer.
- Out-of-order execution and out-of-order completion allows instructions to run efficiently
- The WAW and WAR dependencies (storage conflicts) can be addressed by register renaming, but the RAW and other resource conflicts needed to be considered.