Monday, November 27, 2017

Instruction Issue Policy

- In essence, the processor is trying to **look ahead of current point** of execution to **locate instructions** that can be brought **into the pipeline**.
- One **constraint**: **result must be correct**. So, processor must **accommodate** the various **dependencies and conflicts** discussed earlier.
- Four categories:
    - In-order issue (order to execute), in-order completion (order to write the result)
    - In-order issue, out-of-order completion
    - Out-of-order issue, out-of-order completion
    - Out-of-order issue, in-order completion
- Example:
    - a superscalar pipeline capable of fetching and decoding 2 instructions at a time
        - Instructions are fetched in pair. the next two instructions must wait until the pair of decode pipeline stages has cleared.
    - having 3 separate function units (e.g., two integer arithmetic and one 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 functional unit (e.g., both need floating-point arithmetic)
        - I5 depends on the value produced by I4
        - I5 and I6 conflict for a functional unit
    - 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, in-order completion (simplest)

- **Sequential execution** (in-order issue) and to write results in that same order (in-order completion)
- Note: in this example, I3 and I4 can use ALU2 or ALU3 at cycle 3. But they must be written back after I1 and I2 have been written. So, put them in cycle 3 won't shorten the time.

In-order issue, in-order completion

Decode		ALU1	ALU2	ALU3	Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4	I3		5
	I6		I5		I4		6
			I6		I5		7
					I6		8

In-order issue, out-of-order completion

- *I2 is allowed to run to completion prior to I1. This allows I3 to be completed earlier.* **Save one cycle**.

In-order issue, out-of-order completion

Decode		ALU1	ALU2	ALU3	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

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

With **in-order issue**, the processor will only **decode** instructions **up to the point of dependency or conflict**.
The processor **cannot look ahead of 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, to 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

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

Decode		Window	ALU1	ALU2	ALU3	Write		Cycle
I1	I2							1
I3	I4	I1, I2	I1	I2				2
I5	I6	I1, I3, I4	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

- Since instructions have been decoded, processor can look ahead
- It is possible to issue instruction I6 ahead of I5 (I5 depends on I4, but I6 does not). Save one cycle.
- Note: for simplicity, the window has not set a limit in this example.