
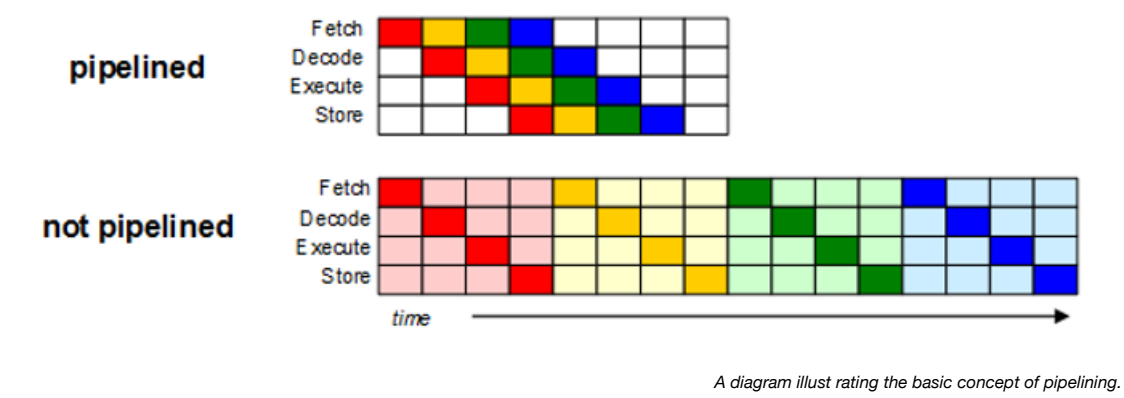


Week 4: Instruction pipelining-2

Instruction pipelining  [. \(http://en.wikipedia.org/wiki/Pipeline_\(computing\)\)](http://en.wikipedia.org/wiki/Pipeline_(computing)) is a technique where the processor works on instructions like an assembly line: with each part of the computation process being split into several ordered segments, so that many instructions could be worked on at the same time (albeit in different stages of execution).




Much like an assembly line, this doesn't make it faster for a single item to be produced (or computed); however it greatly enhances throughput: very handy when there are several million instructions to compute every second!

At its basis, the four main stages of a pipelined CPU are the four execution stages: fetch, decode, execute and store. However, to improve efficiency these stages themselves can be broken up into longer pipelines. These smaller stages are called *micro- ops*.


The Pentium 4 family took the pipelining concept to extremes: it started with a 20-stage pipeline, and by the end of its useful life, the pipeline had no less than 31 stages!

The "length" of a pipeline is known as its depth - e.g. *the Pentium 4 had a very deep pipeline*.

Pipeline stalls

In practice, a pipelined CPU architecture isn't 100% efficient in its throughput of instructions. Be it for reasons of waiting for data from cache or memory, or an unsuccessful branch prediction, these processors can suffer from a **pipeline stall**  [. \(http://en.wikipedia.org/wiki/Bubble_\(computing\)\)](http://en.wikipedia.org/wiki/Bubble_(computing)) - that is, there aren't enough instructions ready to process, such that 'bubbles' of no useful work appear in the pipeline.

Branch prediction

Pipelining has been further enhanced with speculative **branch prediction**  [. \(http://en.wikipedia.org/wiki/Branch_prediction\)](http://en.wikipedia.org/wiki/Branch_prediction): this is where the processor tries to guess what future instructions might be, and will process it in any 'spare time' it might happen to have (for example, inside a pipeline stall). This is a lot more effective than you might think!