Daniel Bishop Jordan Van Duyne
Thuc Tran Isaac Getto

Multi-Core Processing

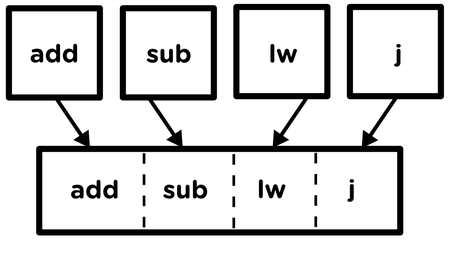
Computer Architecture

Multi-Core Processing

Multi-core processing is a feature common to most modern processors in which a singular CPU contains multiple execution cores. Each core is capable of executing one instruction at a time, allowing the CPU to execute many things simultaneously. Ideally, a CPU with 4 cores would be able to do tasks 4x faster than one with only 1 core. However, problems such as instruction parallelization mean that the speed benefit from multiple cores is often a diminishing return.

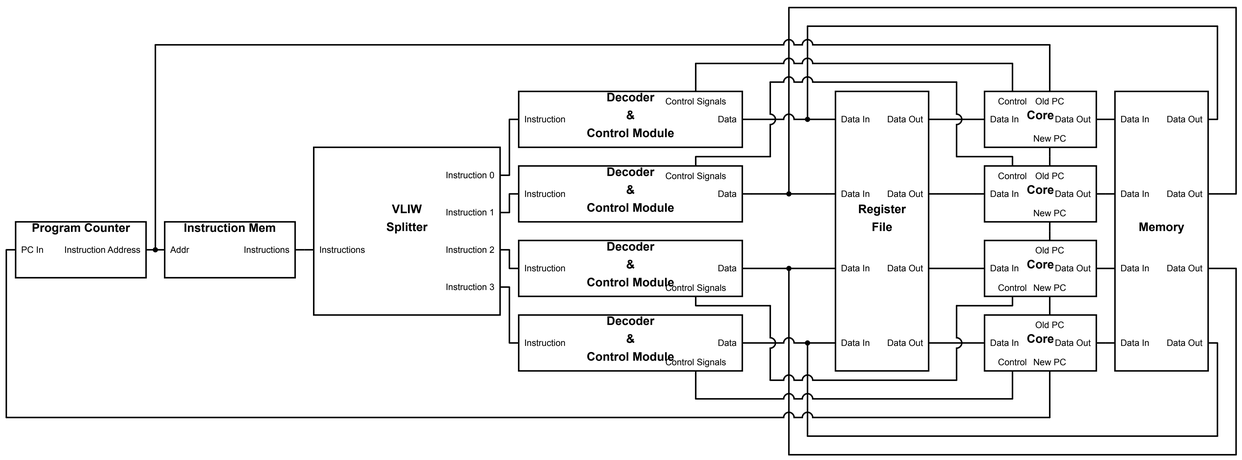
Very Long Instruction Words

Very Long Instruction Words (VLIW) are simply defined as an instruction that contains a set of instructions. For our purposes, we use VLIWs that are just 4 normal instructions stitched together. Each instruction in our VLIW is sent to a corresponding core to be executed. The formatting of the VLIW is up to the compiler, so it is the software designer's job to ensure that conflicting instructions do not appear in the same VLIW.



Architecture

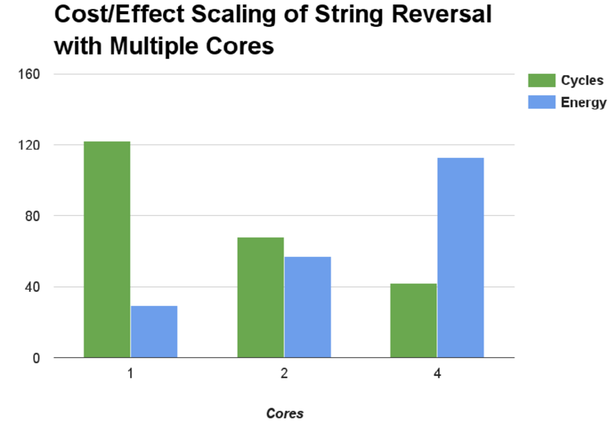
We used a combination of traditional multi-core and VLIW architectures in our processor. Our instruction stream is composed of VLIWs, which then are fed through a splitter and decoder to generate control and data signals for each core. Cores are built around an ALU with some peripheral logic gates. All cores access a shared register file and memory module, making data hazards a threat but allowing instructions to be core-independent.



Strengths/Weaknesses

+ Execute as many instructions simultaneously as you have cores
+ Jumping hierarchy allows for easy branching
+ All registers and memory shared between cores

- VLIW means compiler has to ensure instruction parallelism
- Limited registers to work with, all cores have to share same set
- Branch/jump values must be calculated with VLIW in mind



Expandability

Instruction Scheduling: Most modern CPU's have some sort of hardware scheduling, letting them parse a single instruction stream and detect instruction dependencies. This would be a good next step for our project as it would remove the need for VLIWs and bring us more inline with modern methods.

Private Caching: Having one shared level of memory leads to a lot of data hazards. Modern CPU's get around this by making levels of memory, where each core has its own private memory and mirrors changes to a larger level of shared memory.

Compiler: A custom compiler could optimize an input program for our target multi-core architecture. It would find instruction dependencies and insert no-ops when needed as well as re-order instructions to maximize core usage. Modern VLIW architectures rely on the compiler rather than the hardware to handle data dependencies.

Sources

https://www.cs.cmu.edu/~fp/courses/15213-s07/lectures/27-multicore.pdf
http://superuser.com/questions/214331/what-is-the-difference-between-multicore-and-multiprocessor
https://en.wikipedia.org/wiki/Very_long_instruction_word
https://en.wikipedia.org/wiki/Multi-core_processor