Aaron Adler
6.911 - VLIW
Presentation Write-up
March 13, 2000

# VLIW Overview

VLIW stands for Very Long Instruction Word. VLIW is a method that combines multiple standard instructions into one long instruction word. This word contains instructions that can be executed at the same time on separate chips or different parts of the same chip. This provides explicit parallelism. By using VLIW you enable the compiler, not the chip to determine which instructions can be run concurrently. This is an advantage because the compiler knows more information about the program than the chip does by the time the code gets to the chip.

The ELI-512 was a machine that had VLIW instructions that had 512 standard instructions contained in them. This machine was never actually built, but it implemented trace scheduling which is an important technique in VLIW processing. Trace scheduling is when the compiler processes the code and determines which path is used the most frequently traveled. The compiler then optimizes this path. The basic blocks that compose the path are separated from the other basic blocks. The path is then optimized and rejoined with the other basic blocks. The rejoining includes special split and rejoin blocks that help align the converted code with the original code.

Dynamic scheduling is another important method when compiling VLIW code. The process, called split-issue, splits the code into two phases, phase one and phase two. This allows for multiple instructions to execute at the same time. Thus, instructions that have certain delays associated with them can be run concurrently, and out-of-order execution is possible. Hardware support is needed to implement this technique and requires delay buffers and temporary variable space in the hardware. The temporary variable space is needed to store results when they come in. The results computed in phase two are stored in temporary variables and are loaded into the appropriate phase one register when they are needed.

Transmeta is currently releasing the Crusoe processor. It uses an innovative technique of having a software layer around its hardware core. This software layer is

useful in many ways. It allows bugs in the hardware to be corrected, it allows the software to emulate any instruction set it wishes, and it allows changes in the hardware to be transparent to the user as they see the same interface (the “Code Morphing” software). Currently the software is configured for x86 instructions, but it can also be configured to be a Java VM or something else. The chip optimizes the code on the fly, and keeps track of any self-modifying code, and which code is run the most frequently. It then optimizes the frequently run code as much as it can. If operations are repeated over and over again, the chip optimizes them more than infrequently used instructions. The chip is aimed at lightweight laptop computers, and because of this it has various power modes that allow it to regulate both the processor speed and voltage.

The Crusoe is different from standard processors because it incurs a cost at the beginning of each instruction to optimize the instruction. However, once the instruction is in memory, it is very fast at executing them. The instructions in the Crusoe come in both 2 and 4 “atom” lengths (an atom is a standard instruction). There are varying levels at which the code can be optimized. The chip uses 16MB of standard physical memory for caching the optimized instructions.

The chip also has special shadow registers and a gated store buffer on the chip to facilitate speculatively computing values. The method of speculation varies depending on how often each path is used. If one is used almost all the time, then that path is run, whereas if two paths are equally likely, then both are run. The shadow registers and the gated store buffers allow the instructions to be backed out if they are incorrect. Also, if an exception is thrown, the last set of instructions can be backed out and the execution can be done more conservatively. There are also instructions called “load and protect” and “store under alias” that allow the code morphing software to reorder the load and store instructions.

It is still too early to tell how well the Crusoe will work. The code morphing software seems promising, but the performance of the chip is still unknown. Also, if many programs are being run and the chip is forced to dump its optimized instruction out of memory, the impact on performance is unclear. Another limitation is the 4 instruction types to send to the Crusoe. The limited instructions allow for only one operation of each type to be executed at once, for example, one ALU operation and one store in each

instruction. This is a drawback because the unused slots must be filled with NOPs that limits the parallelism of the chip.

VLIW machines are similar in some respects to vector machines, but they are also different. Vector machines perform the same operation on a vector of data while the VLIW machines perform multiple operations at the same time. Because of this, the VLIW machines have to have many more branch statements in the instructions, which is a key difference. The Crusoe processor looks just like a x86 chip to the BIOS, applications, and operating system of the computer, which is a very nice feature of the chip. It also has other advantages; it runs with less power and energy, and thus with less heat, which in theory may help improve the battery life of laptops.

VLIW code is more efficient if you can have multiple processors running computations at the same time, and if you can keep all the processors busy. The hard part is making sure that all the processors are busy and that there are not too many NOPs in the instructions. If only a few processors are running, then you are not gaining that much over conventional processors, but if you can keep all of them busy, then you will be able to run the code more quickly. The Crusoe seems to be a promising chip, but only time will tell if it brings VLIW computing into mainstream laptops and handheld devices.