**Literature Review #1**

**Introduction**

In this introduction, Patterson touches on the general trend in computers today. He is saying that the complexity of computer architecture is increasing as computers become more and more powerful. He argues that his increased complexity is not a virtue, it is rather unnecessary and causes inefficiencies in the hardware. He then turns to his investigation into the current very large scale integration (VLSI) architectures, and points out their flaws, mainly the delay-power penalty of data transfers between different chips. He is saying that even if you have millions of chips, they are still delimited by the fact that data cannot be transferred between them as efficiently as would be ideal. This fact is what lead to the creation of RISC, says Patterson, and the purpose of the RISC project is to explore alternatives to the increasing complexity of newer computer architectures. He then goes on to hypothesize that by reducing the instruction set, that RISC can make better use of the resources provided by the computer. Also, he predicts that the reduced instruction set will lower design time, the number of design errors, and the execution time of each instruction. Patterson then lists four goals for the project, these include: executing one instruction every cycle, making all instructions the same size, only use load and store keywords to load/store memory, adding support for high-level languages. Logically, these constraints should make for poorer code density and/or poorer performance, but Patterson says the RISC architecture has been competing well, due to their innovative overlapped registers.

**Support for High-Level Languages**

This section starts with Patterson saying that new computer architectures should be designed so that they can work in tandem with high-level languages. Whether or not a high-level language is implemented primarily by the hardware, or software does not matter as long as there is the right level of abstraction from its implementation. He says that the role of any architect is to create a cost-effective system by delegating to the hardware/software when appropriate. He chose C, and Pascal for his implementation, due to their large user communities. RISC architecture is mainly software based, due to the limited number of transistors in single-chip computers, and hardware is used for only the instructions that take the most cycles. He then goes on to investigate C programs, to see what constructs are used most often, and finds that integer constants are just as common as structures and arrays. Also, most structures are local variables, and the vast majority of arrays are globally declared. Patterson then goes on to determining the relative frequency of statements in high-level language programs. After thorough analysis, it was found that call/return statements were the most time-consuming operation in typical programs. Optimizing these inefficiencies is one of the main goals of RISC.

**Basic Architecture of RISC I**

The RISC I instruction set contains simple operations, and these instructions operate on registers, which are 32-bit segments of data. There are four types of instructions in RISC: arithmetic logical (ALU), memory access, branch, and miscellaneous. Each RISC I cycle time is calculated based on the time it takes to do some ALU operation, and store that result back into a register. Contrary to their goals of having each instruction execute in one cycle, load/store instructions must take two full cycles. This is seen as a better alternative to increasing the length of the overall cycle, for only two commands. These memory access commands use “index plus displacement” and “register indirect addressing” to make memory access as efficient as possible. The real innovation of RISC I, comes from the optimization of branch statements, specifically the call, return, and jump subroutines. He then goes on to explain how these optimizations were done. He also states that since integer constants appear to be so common in high-level programs most instruction have an immediate counterpart, that allows for the passing of constants (like addi). He then goes onto state the importance that the call instruction be as fast as possible because it is so common, even more so in RISC, so its optimization is critical to the performance of this architecture. RISC uses something called the *register window scheme* that limits the number of data accesses, to make call instructions faster. To avoid expensive operations like saving and restoring, call procedures in RISC use a new set of registers for each call and just changes pointers between these registers to save time. Also, RISC uses register overlap to allow parameters to be passed directly from register to register, instead of using the stack. The problem may arise that there are too many calls being issued and not enough registers to put the results in, in this case an overflow stack is used to store the overflow, and stack pointer is used to point at this data structure. To make variable allocation faster, RISC allows for pointers to registers, typically pointers to memory would be handled by the compiler. However, this slows down data access, so RISC gave registers each an address, that can be used for pointers, so with just one comparison it can be known if any given pointer points to a register, or some random location in memory. He then goes on to talk about prefetching the next instruction during the execution of the current instruction, this increases performance, but does have problems specifically with regard to branching. Prefetching is generally too complicated for a single chip, so something called a delayed jump was developed. This type of jump allows for the correct instruction to always be prefetched, despite a branch statement.

**Evaluation**

The purpose of this section is to evaluate the register window scheme discussed previously, delayed branching, and the overall performance of RISC I. The register window scheme, after two benchmark tests, has been shown to reduce the cost of procedures. When comparing the procedure mechanism to traditional machines, the procedure cost was still shown to be reduced significantly. This scheme also reduces off-chip memory access, by around 10-30% with 50% of all instructions being register-to-register, which in traditional machines is only around 20%. With regard to delayed branching, it has been shown to reduce the number of no operation instructions (NOP), which decreases unnecessary instructions, and saves on execution time. Next is the overall performance of RISC I, it is being compared against VAX, and PDP-11, both which were mainstays at the time. It was found that RISC has two-thirds more instructions than VAX, and two-fifths more instructions than PDP-11. However, RISC programs were only 50% larger, which was smaller than anticipated. He then goes onto to state the size of programs does not matter as much as the performance. RISC was tested against the competition in different programs, from sorting, to puzzle completing algorithms. It was shown that RISC I was successful in reducing the number of data access is all programs substantially, however the number of words accessed was increased due to inefficient instruction encoding. When it comes to execution time, it is hard to say with RISC, since there is no hardware to test it with, and it seems that RISC, at its capacity during this paper, was not as fast as Patterson would have liked. However, he believed that this could easily be improved with further permutations.

**Memory Interface**

This conclusory section talks about how memory is generally the main bottleneck in computers. Patterson postulates that as intrinsic CPU speed increases, on-board memory (cache) will need to be used to make data access much faster, and less of a bottleneck. Said on-board memory would potentially make RISC much faster, since off-chip data access is a major point of inefficiency in RISC, and other similar architectures.

**Summary**

Patterson summarily judges the efficacy of this new architecture, saying that the reduced instruction set approach to programs is very promising. He also extolls the virtues of the register window scheme, saying that it increases performance. Most of the complexity from modern computers has been taken out with RISC, with little impact on performance. He is hopeful for the future of RISC as it undergoes more permutations and more significant testing.