The Evolution of Computer Technology

Advances in science and technology eventually allowed for the advent of general purpose computers. One could make the argument that computer technology has evolved so much since then, that it aids advances in other fields. To understand how computers have evolved, it will be necessary to first understand some key concepts that. Doing so will allow us to understand how they work together and how they have changed. Specifically, we will examine how reduced instruction set computer architectures (RISC), pipelining, cache, and virtual memory have improved system performance in computer technology. We will investigate each of the aforementioned concepts individually and how advances in their respective scopes have contributed to improvements in system performance.

Generally, microprocessor architectures are divided into two categories, RISC and CISC (Complex instruction set computers.) RISC has a simple and fixed-size instruction set in which only a few of those instructions access memory (Load and Store). Contrast this to CISC, which has a large number of instructions that vary in length. In RISC, one instruction is one operation, which is not the case for CISC either. In addition, CISC has more instructions, which access memory, unlike RISC, which only has two instructions that access memory. Since accessing memory is an expensive operation, it would improve system performance not to access it minimally. This, in addition to effect of simpler instructions are the real advantages of RISC. Of course, simpler instructions does not simply mean superior. In fact, it may take a RISC machine more instructions to perform an instruction than a CISC machine, but this also is not necessarily inferior for performance. The comparative advantage is that since each instruction is simpler, they take less clock cycles than CISC instructions. Additionally, the clock cycles in a RISC machine can be shorter which means an overall increase in speed. The uniformity of one clock cycle per instruction in RISC machines also makes pipelining possible due to the predictability of clock cycles, which further increases system performance.

The simplicity in RISC made room for more improvements in computer systems. It has led to simpler CPU design, which made production of chips cheaper. RISC also requires large banks of registers. This is how RISC machines are able to have less access to main memory – also improving system performance as previously mentioned. Adding registers to computer architectures has had other secondary positive effects on the evolution of computers due to the use of general registers by other components in the system.

Although RISC has had a positive impact on system performance, CISC architectures are still in production and they have advantages of their own as well. Many architectures will implement certain parts of RISC and parts of CISC making the lines of differentiation blurry.

Pipelining became easier with the adoption of RISC architectures and more importantly, improved system performance by greatly reducing the number of clock cycles necessary to run a set of instructions. This is accomplished by completing different parts of an instruction simultaneously. For example, after fetching for the first instruction, the processor can fetch for the next instruction while the first instruction moves onto the next stage. This leads to increased CPU throughput, which leads to overall better system performance. Pipelining

The improvement in efficiency brought by pipelining does not come without possible negative side effects. If a later instruction has a dependency on an earlier instruction, then it may be possible for an unexpected outcome. For example, if there is an arithmetic instruction and then an instruction following it to access the result of the previous instruction, if the result is not yet calculated and stored in the expected location of the second instruction, then the second instruction will not get the value expected. To deal with this, a common solution is to stall an instruction for any number of clock cycles necessary. This ensures accuracy, but is also a realization of some of the limits of pipelining.

For the remainder of this paper, memory and management of memory will be the primary focus, specifically virtual memory and cache. Virtual memory allows many programs to run even though the total memory used by these programs may exceed main memory. There were ways of handling memory management issues prior to virtual memory in the form it takes today, such as overlaying. Overlaying simply split programs up and only loaded what was needed at any given point in time. Compare this to virtual memory that allocates actual hard disk space to create what a program thinks is contiguous memory, even though the program may physically be non-contiguous on RAM due to some other program. Previously, this would have led to issues such as fragmentation where slivers of memory that are no longer being used are too small for any purpose that might be needed by the next or active program.

Virtual memory solves a few problems, which improves user experience like running multiple programs at one time. This also improves system performance, as a single program should not require as many accesses to physical memory as the case may have been with overlays. Programs also do not write over each other as they have access to the virtual memory and do not access physical memory locations directly.

How do programs run in virtual memory? The answer is that they still actually run in their physical memory space and use either segmentation or page tables to map to physical locations. This works by taking a virtual memory address and looking up the physical address that it actually resides on all of which is located on the page table. The more memory addresses there are, the larger the paging table grows which slows the system with each page table look up.