Processor Evolution:
A look at CISC, RISC, and EPIC Architectures

Bradly M. Hussey

Abstract: The purpose of this paper is to provide the reader a basic description of the three most prominent processor architectures. Following the evolutionary order, the CISC architecture will be looked at first. The RISC and EPIC systems will then be studied. The advantages and disadvantages of each architecture will also be discussed using a comparative view.
II. Introduction

When the first microprocessors were introduced, they exhibited very limited functionality. As time went on, the number of functions they could perform increased in response to demand. To implement the greater functionality, the collection of basic instructions the processor used evolved into a very lengthy list. At the same time, the instructions became progressively more complex. The computers that used these processors came to be known as Complex Instruction Set Computers, or CISC. In the 1980’s, a movement toward a smaller, faster, instruction set developed. This movement led to the creation of a Reduced Instruction Set Computer (RISC). The evolutionary progression has continued in the past few years with the introduction of Explicitly Parallel Instruction Computing (EPIC). The purpose of this paper is to provide the reader an understanding of the differences between these three architectures and the advantages and disadvantages of each.

III. CISC

CISC systems are capable of carrying out many advanced operations through the execution of only one instruction. These processors attempt to increase computational speeds by supplying a broad range of specialized instructions, each of which is designed for specific operations. The premise this architecture is built upon is really quite simple: fewer instruction executions for a task mean fewer clock cycles to complete the task.

One of the greatest advantages of the CISC architecture is that complex tasks can be performed with relatively small amounts of program code. This means that overall system performance is high because very few distinct instructions must be processed to reach an end result (Gerritsen, 2002). This also makes the programmer’s job easier as they are able to use more complex programming structures which are the high-level equivalents to the included complex instructions (Patterson, 1982) (Dietzel, 1980). Another advantage is that by natively supporting many advanced operations, there is a diminished need for storing temporary data in main memory during very complex operations, and with smaller programs, fewer lines of code need to be placed in memory and referenced (Carpinelli, 2001). Accessing memory to store or retrieve data is a very costly transaction in terms of time. Another nice aspect of CISC architecture is the

---

1 The term CISC was retroactively introduced after the term RISC was adopted to make a distinction.
2 There is, of course, the corollary that CISC chips use a slower clock speed.
backwards compatibility built into successive generations of processors. Because CISC is more concerned with a smaller number of instructions to complete the task, designers do not remove the previous generation’s instruction set—they simply add new instructions. This makes consumers very happy since they do not need to replace all of their software with every new processor generation (Dileep, 1991). Lastly, because CISC uses more complex instructions, there is a smaller gap between high-level languages and the instructions, which means a much faster and simpler compiler can be used.

There is, however, a downside to the CISC philosophy. CISC systems are inherently slow. Instructions that are more complex take longer to execute than do simple instructions. For example, the multiplication function will take longer to process than simple addition does. CISC technology also has a longer propagation delay because the decoder used to generate output needs to be very large to accommodate the large number of potential operations that could have been preformed (Carpinelli, 2001). In addition, with every successive generation chip, the number of amassed instructions increases thereby fueling the propagation delay. Speed is not the only drawback though; another disadvantage to CISC architecture is physical size. Including the hardware required for all the instructions leaves very little room for fast, temporary storage registers (De Gelas, 2002).

IV. RISC

RISC systems were developed in the 1980’s as an alternative to the conventional CISC architecture. This system, as the name implies, uses a relatively small number of simple instructions to perform operations. Using a small set of fixed-length instructions, forces the chip designer to use only simple “precursor” instructions, which can be recombined in different ways to mimic a more complex instruction. The RISC processor is based on the fact that the average program utilizes less than twenty percent of the available instructions supported by a CISC processor (Amiga Interactive Guide, 2002). Each of these unused instructions serves only to slow down the overall performance. Hence removing the unused, complex instructions will likely lead to performance increases.

The strongest argument for the RISC system is that each simple instruction can be executed much faster than a complex instruction. RISC chips also have a much shorter propagation delay due to the fact that smaller decoders can be used for generating output. These two variables mean that a faster clock cycle
can be used to perform single operations very rapidly (Gerritsen, 2002). Another advantage is that RISC processors, because of their limited instruction set, require very little hardware for carrying out operations. This leaves a larger area available for adding devices that can further speed up processing, such as high-speed cache memory, pipelining hardware, additional registers, and superscalar support (Gerritsen, 2002). 3

As with the CISC systems, there are some very clear disadvantages found in RISC architecture. The most prominent problem is that a large number of distinct instructions must be executed to carry out a single complex operation. Even though each individual operation is performed relatively fast, the total amount of time required can be very high. Using smaller, simple instructions leads to larger applications. The number of lines of code required for complex programming structures is greatly increased because several lines for precursor “units” must be written to build the whole structure. Not only do these added lines take more time to process, they also can cause the application to become large enough for portions of it to be stored on disk or in memory, which are very inefficient storage locations (Amiga Interactive Guide, 2002) (Carpinelli, 2001). Also, when programming in a high-level language the gap between code and instructions is large which necessitates the use of slower, more complex compilers. Another strike against RISC processors is the lack of legacy support. Lastly, it is interesting to note that the advantage of superscalar processing is also viewed as a disadvantage. Many new RISC chips incorporate hardware to determine which instructions can be executed in parallel. Frequently, there are long stretches when parallel instruction execution cannot be used, which means time is being wasted while the processor is trying to find an optimized, parallel-execution solution (Love, 1990).

V. EPIC

The next rung on the evolutionary ladder is the EPIC architecture. EPIC is a system with very strong roots in its predecessor. In fact, many experts would argue that EPIC is no more than an extension of RISC (Every, 2002). 4 Regardless of this argument, the goal of EPIC is increased system performance through parallel execution of instructions, predication, speculative execution, and better handling of branches.

---

3 Most of these devices were gradually added throughout the evolution of RISC, and were not a part of the original design.

4 To be precise, it is often considered an extension of the Very-Long Instruction-Word (VLIW) variant form of RISC, which will not be covered here.
The key advantage to EPIC is stated right in the name—explicitly parallel. EPIC places the burden of deciding which instructions can be executed in parallel on the compiler. This allows the processor to spend its time executing instructions rather than guessing at which instructions might be able to execute simultaneously as is the case with RISC’s superscalar (De Gelas, 2002) (Love, 1990). EPIC also loads bundles of instructions for parallel execution rather than single instructions, which allows for a higher degree of parallelism (De Gelas, 2002). Predication is another feature EPIC systems display, and it is a very simple concept that saves time. At the same time as the processor is executing an instruction, it is also checking to ensure that the instruction should be performed. If the processor finds that it should not have done the operation, no results are stored and the processor skips ahead to the next instruction. Although newer RISC chips have a similar notion called annulment, predication is proactive while annulment is reactive (August, 1998) (Driesen, 1998). This small difference in the order of operations is enough to make a positive impact on system performance. A highly related feature of EPIC obsoletes the branch prediction function found in RISC systems. Whereas RISC always predicted that a clause was true, thus loading the “if branch” into the pipeline, EPIC simultaneously follows through both branches of the condition and only records the correct one once it is determined (Driesen, 1998). In a RISC system, if the program should have followed the non-default path, the processor often needed to roll back some changes, empty the pipeline and then start over, but this time taking the correct path. The last, truly significant advantage the EPIC architecture shows is speculative execution. Speculative execution is the process whereby the processor reads ahead in the program and loads data into cache before it is actually needed, much like pre-fetching instructions. If the program reaches the point where the data is needed, the storage register will send its stored information; if the “future” instruction is skipped, the cached information is simply overwritten. Performing this simple pre-load saves considerable time by allowing the processor to avoid memory access while executing instructions. (De Gelas, 2002)

Although EPIC may initially sound like the answer to all existing problems, this architecture has its faults too. The core of EPIC’s performance boost is based on the compiler’s ability to create a static instruction schedule with a high degree of parallelism. This is not always possible because often variables are introduced at runtime which are not known at compile time, especially in object oriented languages (Love, 1990). Another potential problem arises when compiling programs that rely on shared libraries.
The compiler does not have access to the library code, which makes it impossible for the compiler to make a complete instruction schedule. (Every, 2002)

VI. Conclusion

Clearly, one can see that in the relatively short span of time computers have been around, significant modifications, to the fundamental architecture, have been implemented. After years of simply adding more instruction to a processor, it came out that the CISC model was losing speed due to the highly customized (not reusable), complex instruction set. RISC designers decided to drastically cut the number of available instructions, thus increasing processor performance. Several years later, the EPIC architecture was developed. The creation of this architecture was partially in response to the limited ability of RISC’s parallel instruction execution. The EPIC architecture, once again, pushed the benchmark forward. This cycle of improving on existing architectures is likely to continue into the foreseeable future. Emerging fields like biological and quantum computing are on the horizon, but far from realization.
Works Cited


Dileep, B. & Clark, D., Performance from Architecture: Comparing a RISC and a CISC with similar Hardware Organizations, Proceedings of the fourth international conference on Architectural support for programming languages and operating systems, 1991

Ditzel, D. & Patterson, D., Retrospective on high-level language computer architecture, International Conference on Computer Architecture Conference proceedings of the seventh annual symposium on Computer Architecture 1980 , La Baule, United States


