Components of a Computer Architecture Education

Yale N. Patt
University of Michigan
Ann Arbor, MI

Abstract

Computer Architecture, if it is a science, is a science of tradeoffs. Thus, the more cases (computer designs) we examine and the more we understand about the components of a particular case, the better we can make those trade-offs. The reality, of course, is that computer architecture is not a science at all, but rather a craft. This paper explores some of the training that is essential in developing the capability of the craftsman.

1 Introduction

This paper represents my views as to what a computer architecture education should consist of. I will describe a set of courses which I suggest constitutes a core computer architecture curriculum. I will also offer some statements about what I think are important to learn in some of those courses. First, I feel compelled to make a few introductory remarks. My view is that if you understand some of my prejudices up front, you are better able to know whether to embrace or disregard all that follows.

First, it should be clear that this treatment is very much a personal note. What I am writing here is based on almost 30 years in the business, – the business of designing and the business of teaching. My views have been influenced by anecdotal evidence which hardly qualifies as a set of empirical laws. However, where the more judicious choose not to tread, I bulldoze forward, having warned the reader that much of what I write here is the result of my experiences practicing my craft, as teacher and designer. I make no pretense that this is systematic science.

Where does computer architecture fit within the spectrum of computing activity? Computer architecture involves the ISA. The ISA (instruction set architecture) is a contract between the compiler writer and the microarchitect, so the better one understands what the compiler can do and what the microarchitecture can do, the more likely one is able to do serious work in computer architecture.

For years, computer science education has started with high level programming. In my view, that is wrong. All good science starts with the foundations, and then builds. Only non-sciences seem to start at the top without proper underpinnings. In that spirit, I argue for a curriculum that goes bottom up. That means the student first learns digital logic design, then computer organization, then actually designs a processor, and lastly treats the architectural issues. My belief is that students who have worked their way up through the trenches are better prepared to discuss the alternatives of the various architectural paradigms.

Computers exist to execute programs, so an understanding of the issues of computer architecture is improved if one understands the nature of compilers and the nature of operating systems. Programs do get compiled before execution, so what a compiler can do and can't do has major impact on the execution of that program. The operating system occupies the processor an inordinate amount of the time, so knowing its characteristics enhance one's understanding of the demands on a computer architecture.

Processors today are implemented on single chips consisting of many millions of transistors. The Digital Alpha 21164, for example, boasts 9.3 million transistors on a single piece of silicon. Consequently, an understanding of VLSI circuits enhances one's understanding of computer architecture.

CAD, on the other hand, is like a sliderule. That is, CAD provides a set of tools such that no serious computer designer would venture forth without a good set. But, like a slide rule, it is necessary that the computer designer understand the internal structure of the tools only to the extent that is necessary to use them effectively.

Computer architecture is not a theoretical discipline, although an appreciation of theory is usually helpful. That is, although a computer is not a Petri net, nor a Universal Turing Machine, the ability to examine processing capability from the standpoint of those conceptual models can provide useful insights. Nonetheless, one must always keep in mind that at the bottom line, computer architecture involves real machines doing real work. Real machines have "gotchas," and you don't master computer architecture without exposure to the complexities inherent in all real machines.

Industry can be immensely important in teaching a student computer architecture because industry builds real machines. I would insist that every student of computer architecture be required to couple periods of on-the-job training in industry with the curriculum he/she absorbs in the university.

2 The Core Curriculum

With the biases stated above, I would argue for a computer architecture core curriculum consisting of four essential courses, preceded by a universal first course in computer science and engineering (Course

0). These four courses (Courses 1 through 4) represent the computer architecture component of a larger Computer Science and/or Engineering Curriculum. Other than Course 0, the non-computer-architecture courses are not discussed below. The curriculum is as follows:

2.1 Introduction to the Discipline

The first course, not specific to computer architecture, but rather to the entire computer science and engineering curriculum, should deal with fundamentals. The student should come away understanding the basics of how a computer processes information, what is a program counter, the fetch, decode, etc. instruction cycle, the notion of memory and the difference between a memory location's address and the value stored at that address. The student should come away with a basic understanding of the elements that make up a data path, and the gates that combine to form elements of that data path.

The student should also come away with a basic understanding of the elements of a programming language and the notion of translation from programs written in that language to the executable image that is presented to the microarchitecture. The student should come away with some facility in writing programs in both the ISA that directly interfaces with the microarchitecture and in a high level language that has to be translated first into the ISA before it can be executed. The student should come away with a basic understanding of the execution time of a program so as to make choices with respect to what is a good algorithm or a bad algorithm.

In short, the first course should be a bottom up treatment of the various elements of a computer that come together to do useful work. Where possible, nothing should be left to "magic." This first course gives the overall picture, from which the following four courses then build the discipline of the core of computer architecture. I would add parenthetically that this overall picture also enables the building of the other sub-disciplines of computer science and engineering.

2.2 Course 1: A serious digital logic design course

A serious computer design course provides the student with the experience of dealing with the tradeoffs that go into producing one complete computer design. In this course students are given the specification of an ISA they are to implement; how they do it is up to them. The design component of this course is (obviously) very high. Their designs should be complete, down to the gates, using a CAD system that requires students to analyze their designs, to perform timing analyses, and to execute code on their finished products. Designs should implement real ISAs, with the gotchas of a specific real-world machine. It does not much matter which real world machine one designs, they all have their own ideosyncracies.

In the real world, when elegance is at cross-purposes with increased performance, elegance loses. Therefore, my preference is a subset of a serious commercial ISA. In the past, I have personally used subsets of the Intel 486, Digital's Alpha, and Hewlett-Packard's Precision Architecture, for example. Whether the student's design is microcoded or hardwired, whether it is heavily pipelined or not matters a lot less than the fact that the student takes the design all the way down to debugged gates.

2.3 Course 4: Comparative computer architecture

Once students understand the fundamentals, have facility with the building blocks, and have done a serious design project, they are ready to test the implications of various architectural choices. For example, the effects of varying the characteristics of the cache (size, associativity, block size, etc.), the effects of speculative execution, the nature of the instruction set architecture, insterrupt structure, I/O handling, etc. all make more sense after Courses 1,2, and 3 above.

2.4 Beyond the core

This concludes what I believe is essential to a core curriculum in computer architecture. Thus it represents the end of the foundation, not the end of the story. One could easily augment this core with courses on multiprocessors, vector processing, I/O architecture, fault tolerant computers, etc. These, I agree, are useful, but they are all beyond the core, and would not be appropriate substitutions in my view for the courses listed above.

3 Concluding Remarks

These are exciting times for computer architecture.

First of all, the field continues to evolve. IBM took a major departure from their very visible 360/370 line less than ten years ago, with the introduction of the RS/6000. Since then, they have joined forces with Motorola to specify still another ISA, the Power-PC. Digital announced a major departure from their long-standing VAX line less than three years ago. Rumor has it that Intel and HP will introduce a new architecture within the next two years that will represent a departure from the most profitable ISA in the history of computing, Intel's x86. Less dramatic, but still exciting, are the continuing changes to the other architectures such as SPARC (v9) and HP PA.

Also, quite apart from the need for practicing computer architects, there is a growing awareness that algoritms work better (i.e., execute faster) if their authors have a clue as to the nature of the hardware these algorithms are to run on. You can easily throw away all the performance a modern processor buys you, if you don't know how to harness that horsepower. Ergo, greater attention is being paid to computer architecture by the algorithm community. For every professional in the computer architecture community, there is a large number in the algorithm community.

So. there is new knowledge to impart, and a growing cadre of professionals who need that new knowledge. How we go about training these professionals will say a lot about how well we satisfy that need.