Tradition and Change: What Should We Be Teaching In Computer Architecture?

Daniel P. Siewiorek
Buhl Professor of Electrical and Computer Engineering
and Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Abstract

The structure of computers has been taught for almost 30 years. During that time there have been many changes both in industry and academia. Even amidst all this changed there have been certain invariant challenges to educators. By taking a historical perspective, this paper identifies some of those invariance. The paper concludes with an example of a possible computer architecture course based upon interdisciplinary design that combines both educational and research experiences.

1 Background

Over the past decade the computer industry has undergone major restructuring, from an industry that was once dominated by a small number of vertically-integrated companies to an industry with hundreds of companies each occupying small horizontal niches. Thirty years ago, large mainframe and minicomputer manufacturers develop and supply technologies at all the various levels of abstraction in a computer system from the basic underlying circuits to the processor/memory/input-output hardware to the operating system and the end-user application. These vertically-integrated companies which thrived on specialization preferred to hire students who had deep knowledge about one subject.

Now the industry has become much more fragmented with the various layers in the computer hierarchy provided not only by different companies but also by companies that are geographically distributed. Interfaces between the various layers have become the focus of attention and interoperability the major concern. Today, companies prefer generalists who are able to converse among all levels of the computer hierarchy.

At the same time there have been dramatic changes in universities. Students have been exposed to computers at an earlier age and arrive better prepared with respect to computer technology than ever before. As we have learned how to teach computer architecture there are better educational materials such as textbooks and software. Rapid change in hardware technology which increase complexity at the rate of 10capability and lower cost. Indeed, many of the personal computers that students use today have features that were reserved for supercomputers only 20 years ago. Thus more concepts are required to understand contemporary computer systems.

Design has returned as a major component in undergraduate education. Students are learning that there are many good answers not just a single answer. This complicates the evaluation process since each individual design has to be understood on its own merits. Design is taught best in a studio-like arrangement as opposed to the engineering class lecture style courses rising from the period when engineering education focused on analysis rather than synthesis.

As design has become more complex, a single student cannot complete a realistic design in a single course. Students must operate in teams in order to complete a design. Furthermore, tools have become more important as a means of leveraging student effort. Not only are tools required to handle the complexity of the design they also represent a realistic environment in that upon graduation students would be expected to use tools in industry. However, the complexity of tools has grown as fast as the artifact which they were intended to design. It can easily take an entire course in order to learn how to use an industrial strength tool leaving no time to experience design. On the other hand, university tool sets can be simple and direct however they do not prepare students for the real world. Universities are struggling with mechanisms for handling design and tool complexity. Perhaps history provides some insights on how to handle these trends.

2 Trends in Computer Architecture Education

There are four inter-relating factors in computer architecture education: theory, design, evaluation, and the student. Let's examine the evolution of these four components over the past thirty years.

Theory. Initial efforts at teaching computer architecture attempted to isolate individual principles (such as addressing modes, memory hierarchy, buses, etc.) and then that provide a set of alternatives for implementing the principle. In an attempt to integrate the principles, case studies were introduced. Since there were few well documented case studies, it was difficult to introduce new principles. Next it became common to integrate all the principles into an example general-purpose instruction set. These "teaching examples" fit the various principles together in a precise fashion. While these examples did integrate the principles for the students, the students were often left with the mistaken impression that there was only one way to fit all the pieces together.

Design. Computer architecture is an active discipline. One cannot learn computer architecture without attempting to do design. Initially designs were done with paper and pencil. Subsequently, logical laboratories became available in which gates could be interconnected to form small pieces of a computer such as an arithmetic logic unit. In the early 1970s Digital Equipment Corporation introduced a product called the Register Transfer Modules (RTM) which were also sold as the PDP-16. Figure 1 shows a simple RTM system for summing the integers from 1 to 10. The right hand side of the diagram is composed of the data and memory modules while the left hand side is composed of a string of control modules. The control modules pass an activation signal to the next control module thereby imposing sequencing. Each register transfer was composed of a simple right hand side, left hand side sequence. For example, loading the number N from a switch register into an internal register I would proceed as follows. The student would wire the signal "N put onto the bus" and "I takes off the bus" to the output of the control module. When that output was activated, the "N puts on the bus" pin, realizing that it was the right hand side of the register transfer operation, would put its contents onto the bus and issue to the bus sense module a "data ready" signal. The "I takes from the bus" signal would realize that it was selected and wait to see the data ready signal before taking the data from the bus. When the data was taken from the bus the "I takes from the bus" signal would notify the bus sense module that the operation was completed by a "data accept" signal. When the bus sense module has seen both a data ready and data accept sequence it issues to all the control modules a "done" signal. When the control module which started the operation of reading N into I sees the "done" signal it passes control on to the next module in sequence. Thus register transfer modules made it very easy to design and implement arbitrary algorithms. As a matter of fact it became possible to design a simple computer on a single page as shown in Figure 2. During the late 1970s simulation packages, first made available through universities and ultimately available commercially, replaced hardware as the design vehicle. While more complex designs could be handled, students lost some of the previously-acquired skills such as hardware testing and debugging. More recently, the availability of field-programmable gate arrays (FPGA) and VLSI chip sets (e.g., processors, memory, and input-output) again make it feasible for students to carry designs completely through to hardware.

Evaluation. The simple general-purpose computer has many pedagogical advantages. It is a complete system so that students have a sense of accom-

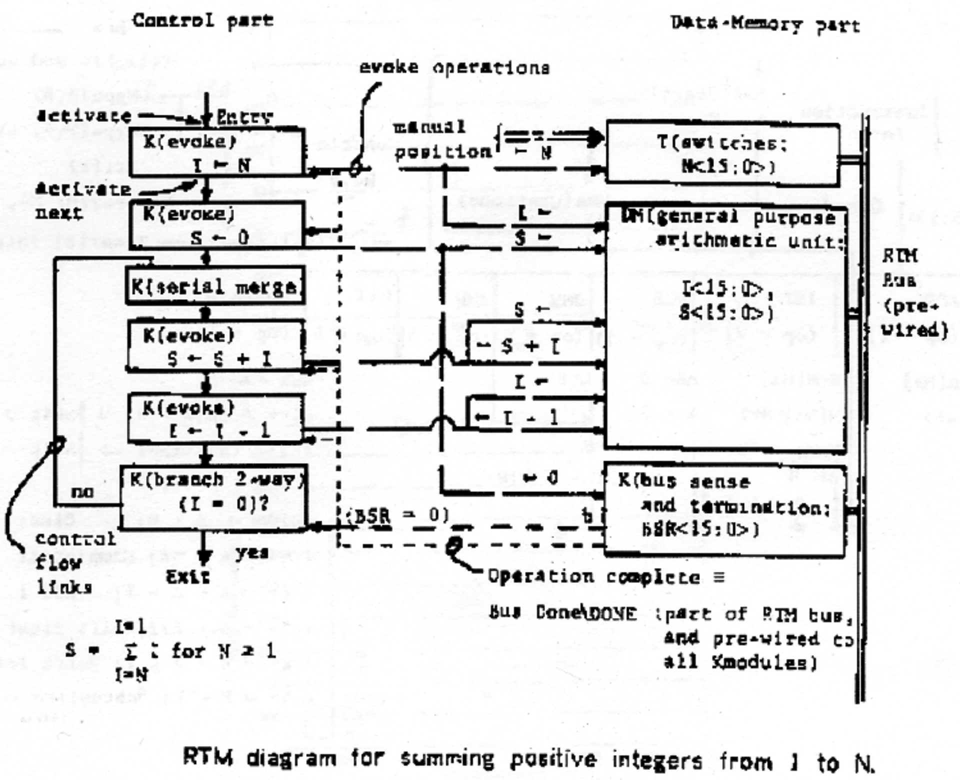

Figure 1: RTM diagram for summing positive integers from 1 to N.

plishment once they finish the design. It incorporates many of the principles in computer architecture education. Furthermore, with a suitable subsetting of the instruction set it is possible to adjust the complexity of the design so that it may be completed during a single course. How can the instructor evaluate the quality of the designs produced by the students? Initially we would calculate the average number of clock cycles to execute all of the instructions in the instruction set. Later it became possible to simulate small benchmarks. Often students get the impression from text books that a 10% increase in performance is an important achievement. Thus after each lab we provide students feedback on the design space that they and their fellow students have explored. One example design space is shown in Figure 6. The data represents student efforts to code a particular benchmark in two instruction sets. The students quickly learned that factors of five-to-one are common between programmers. Furthermore, these programmers may use the instruction sets in ways such that design principles intended by the architect are violated. For example in the plot on the bottom of Figure 6 there are nearly as many cases when the complex instruction set (VAX) requires more instruction executions than the simple instruction set (Alpha) and vice versa.

Students. Instructors and students only have a finite level of effort that they can place on any given course. From the student's side, there is a "rule of constant complexity". The students can only produce a fixed quantum of effort. The goal is to make that effort as meaningful as possible and not to squander it on details about software tools and approaches that havea limited lasting value. Students do very well in designing by analogy. Thus if students are given example test programs and component libraries from which to assemble their designs, they will not be caught in the pitfall of not even knowing how to start. Furthermore,

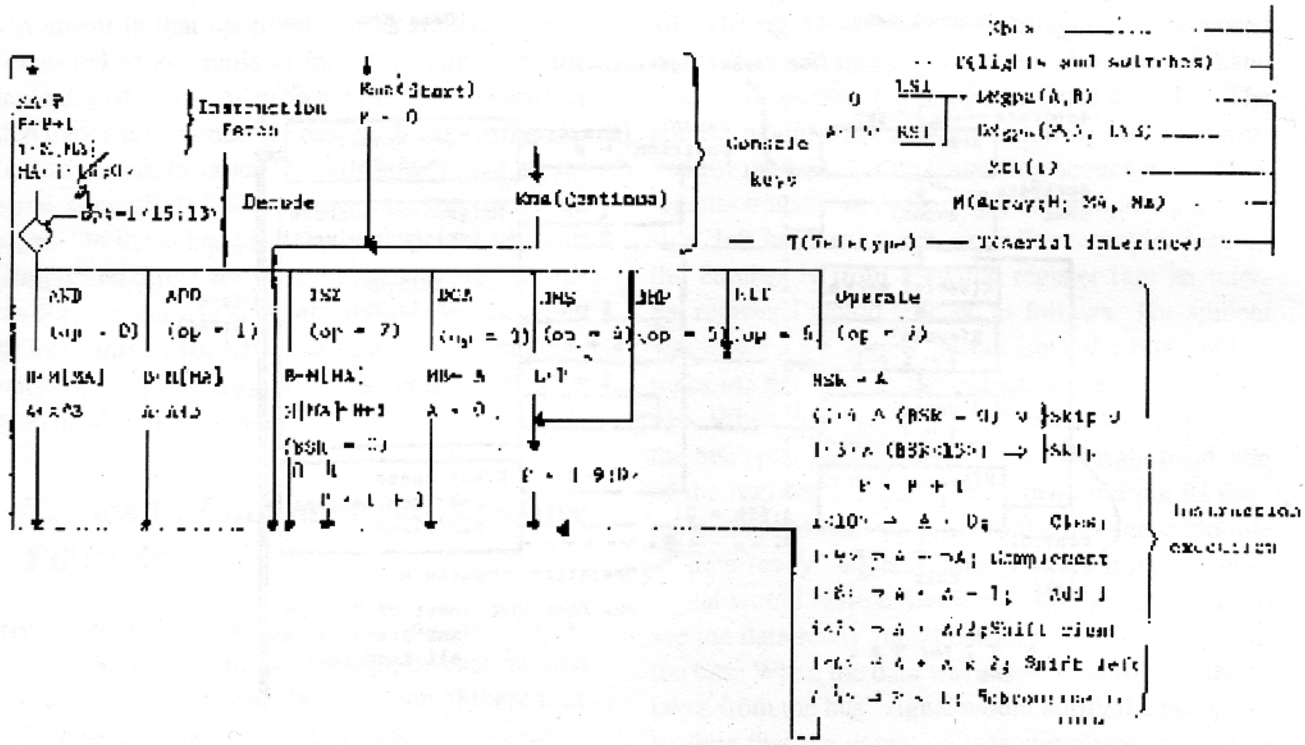

Figure 2: RTM diagram for a simple computer.

letting the students work together as teams also helps minimize wasted effort in looking for answers to fundamental questions during the start-up transition. By talking among themselves students can often figure out an approach to a new topic rather than waiting for office hours or lectures to ask questions. The instructor effort must also be leveraged. In order to simulate an entire benchmark, the complete design had to work correctly. This "all or nothing" approach provided harsh penalties for those students who were very often were within one "bug" of a completely functional system. Thus we have used test scripts which exercise well-defined boundaries and interfaces in the system. By defining critical interfaces such as between the control part and the data part of the system, it becomes possible to initially probe the data part to assure its correct functionality before exploring the control portion. Partial credit can be assigned for these unique designs. Given these trends in the various factors of computer architecture education, the question remains what should be taught in computer architecture for the future.

3 What is the Goal of Computer Architecture Education?

Most computer architecture courses are taught as if the students would become instruction set architecture designers. Less than 1 percent of the students will be involved in design of an instruction set. As noted before, instruction set architectures have been easy to understand while embodying all the major architectural issues. From our rule of "constant complexity" instructors have sought the "one page" description of an instruction set. The description could be textual or represented by a state machine. Figure 2 depicted the state machine of a simplified PDP-8 [1] while Fig-

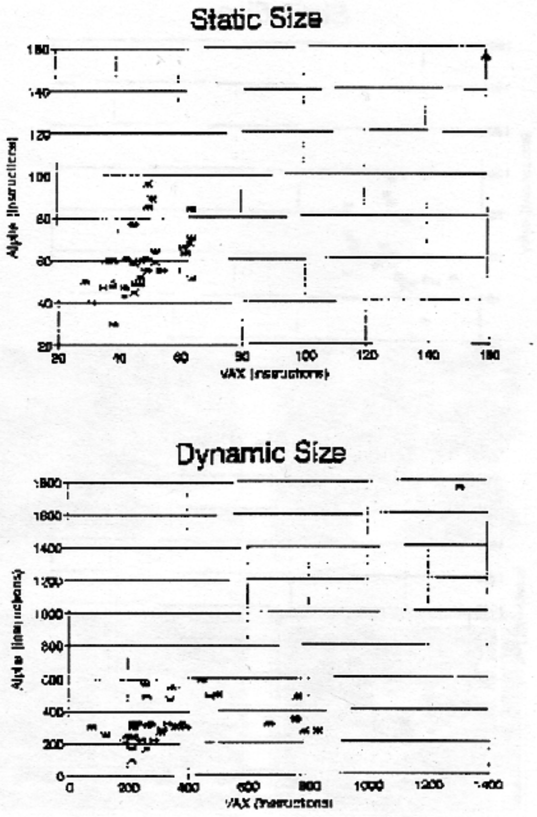

Figure 3: Student effort to code benchmarks in two instruction sets.

ure 4 depicts a subset of the state machine for the DLX [2]. There is a striking similarity between these descriptions which were generated over 20 years apart. There is an instruction fetch, instruction decode, and execution.

A larger portion of the students might design non-instruction set systems to which the principles of locality, concurrency, etc., that they have learned in the instruction set architecture design can be applied. But again, this would represent only a few percent of the student population. By far, the vast majority of the students would become instruction set architecture users wherein they would build systems. Thus in the future we should be educating systems architects.

4 A System Architecture Course

The goal of a system architecture course should be to teach students how to specify systems. In the design of these systems they should use new generation computer-aided design tools. The course should involve multiple disciplines. As an example, consider the wearable computer course taught at Carnegie Mellon University for the past three years. The course involves undergraduates from several disciplines in the design, fabrication, and evaluation of a complete computer system in a four-month semester course. Students include industrial designers, electrical engineers, mechanical engineer, human factors, and computer scientists. The students follow a design methodology that allows as much concurrency as possible during the design process [3]. Concurrency occurs in both time and resources. Time is divided into phases. Activities within a phase proceed in parallel, but are synchronized so that phase boundaries culminate in an integrated demonstration. Resources consists of personnel, hardware platforms, and communications. Personnel resources are dynamically allocated

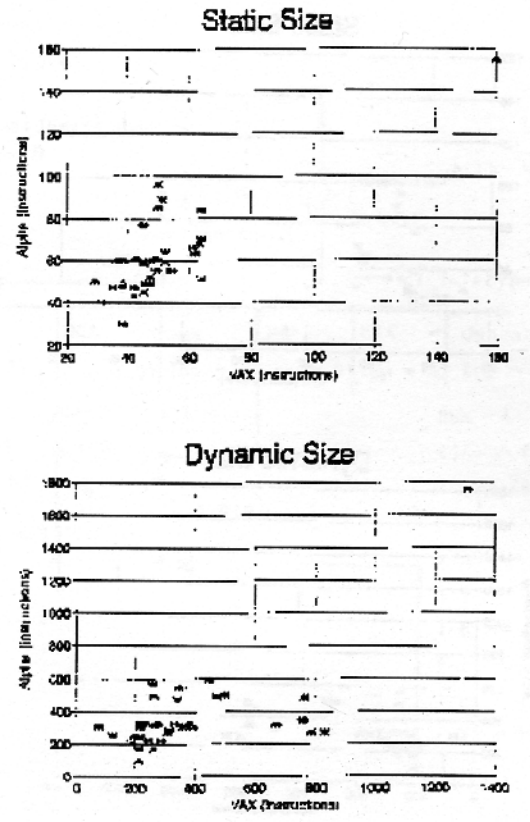

Figure 4: Subset of the state machine for DLX.

to groups which focus on specific problems. Hardware development platforms include workstations for initial design, personal computers for development, and the final target system. Communications allow design groups and individuals to communicate between the synchronization points. The final system is evaluated in field trials.

Figure 5 depicts some of the phases in the design of an initial inspection check list computer for heavy vehicle maintenance. Not only do students gain experience in operating in interdisciplinary design teams, they have created a network of over forty suppliers of components and manufacturing services required for the final system. They learn about interdisciplinary dependencies, scheduling, risk reduction, and how to appreciate different disciplinary views of a single artifact. The systems produced by the students are part of a research project not only identifying how to perform better interdisciplinary design but also experimentally-determining the best form factor for new generations of computers. Students become highly motivated when they realize that not only will their design be fabricated but there will also be users who will depend upon their design to accomplish their work.

[1] Siewiorek, Daniel P., C. Gordon Bell, and Allen Newell, *Computer Structures: Principles and Examples* McGraw-Hill Book Company, NY, NY, 1982.

[2] Hennessy, John L. and David A. Patterson, *Computer Architecture A Quantative Approach,* Morgan Kaufman Publishers, Inc. Palo Alto, CA, 1990.

[3] Siewiorek, Daniel P., Asim Smailagic, Jason C.Y. Lee, Ali Reza Adl-Tabatabai, *An Interdisciplinary Concurrent Design Methodology as Applied to The Navigator Wearable Computer System,* Journal of Computer and Software Engineering, Special Issue on Hardware/Software Co-Design, May 1994.

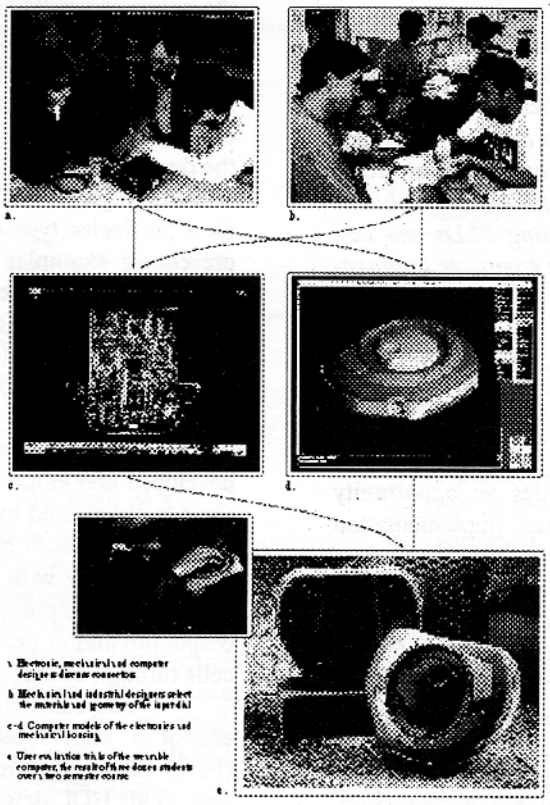
a.
b.
c.
d.
e.
a. Electronic, mechanical and computer designers discuss connectors
b. Mechanical and industrial designers select the materials and geometry of the input dial
c-d. Computer models of the electronics and mechanical housing
e. User evaluation trials of the wearable computer, the result of three dozen students over a two semester course

Figure 5: Phases in the design of a wearable computer.