Teaching the Introductory Computer Architecture Course with a Systematic View

Wei Zhang

Department of Electrical and Computer Engineering
Southern Illinois University Carbondale
Carbondale, IL 62902
USA
zhang@engr.siu.edu

Teaching the Introductory Computer Architecture Course with a Systematic View

Wei Zhang

Department of Electrical and Computer Engineering
Southern Illinois University Carbondale
Carbondale, IL 62902
USA
zhang@engr.siu.edu

Abstract

The introductory courses in computer architecture typically introduce undergraduate students a large number of hardware components and their organizations, including the datapath, control unit, cache, memory, hard disk, bus, other I/O devices, etc. Without a global picture of the computer as a system, students often have difficulties in relating these topics to what they have learned in lower level courses, and can be easily overwhelmed by a large variety of discrete topics, which will impact their learning outcome. This paper presents a new approach to teaching the introductory computer architecture courses with an explicit emphasis on the systematic picture of the computer system. Introducing the high-level framework of computer as a system can enhance students' understanding of various architectural components, and mitigate the difficult of performing hardware design or assembly programming projects on specific architecture topics. In addition, we also highlight the importance of software and its interaction with the underlying hardware by introducing a set of MIPS assembly programming projects. Based on our experience in two subsequent semesters, such an approach can enhance the instruction of the introductory computer architecture course and help students improve their learning outcome.

1. Introduction

Most computer science and computer engineering programs have two or more computer architecture courses [4]. The introductory computer architecture course typically follows a programming course and a logic design course, which is often offered to sophomore or junior students. The goal of the first computer architecture course is generally to provide a basic introduction to the organization and input/output interface of a simple general-purpose microprocessor. More advanced architecture concepts and optimizations are usually provided in the secondary computer architecture course.

This paper is based on the author's experiences in teaching, updating and enhancing the computer architecture courses in a public university in the Midwest of US. The introductory computer architecture course is offered to the junior students in the computer engineering program. The content of the first computer architecture course typically includes the introduction of a number of hardware components of general-purpose microprocessors, such as the datapath, control unit, cache, memory, hard disk, bus, other I/O devices, etc. Typically, the theme of the introductory computer architecture course is to maximize performance. Accordingly, traditional approach to teaching the introductory computer architecture course often begins with an outline of topics, and then introduces the instruction set architecture (e.g., MIPS), the arithmetic for computers, the processor and pipeline, the memory hierarchy and I/O devices [1]. While it is necessary and beneficial to explain the design and implementation of each hardware components thoroughly, the lack of a global view of microprocessors can easily make students lost in the details of discrete topics and may become frustrated by the complexity of the microprocessor. Especially for undergraduate students who have just learned the logic design and basic programming, it is not uncommon for them to ask what is the relationship between computer architecture with what they have learned in other courses such as digital design and basic software programming courses, especially when they are exposed to the architecture of microprocessors for the first time. Moreover, based on students' feedback, without keeping a high-level framework of computer hardware and software and their interactions in mind, students often have difficulties in starting doing simulation or programming projects in the early stage of the course due to the complexity of the computer system, which may delay the progress of the course and impact student learning outcome.

To address the abovementioned problems, this paper argues that it is crucial to explicitly lay emphasis on the systematic view in teaching the introductory computer architecture courses, which emphasizes the high-level organization of the computer systems and the interactions among different hardware components before teaching students the details of each hardware component. A systematic view of computer architecture can not only help students relate different topics that they have learned in preliminary or other courses in the context of computer architecture, but also smoothen their learning process of understanding a variety of individual computer architectural components. Moreover, in our updated computer architecture course, we also stress the importance of software and its interaction with the underlying hardware by introducing a set of assembly programming projects. Our evaluation indicates that teaching introductory computer architecture course with a systematic view can adequately improve student learning outcome.

2. A Systematic View of Computer Systems

A computer system is an integrated system, including a number of computer hardware components, their interconnections, the system and application software, and the interactions between software and underlying hardware. Introducing such a complex system to the students in one course can be a daunting task. The author has taught the introductory computer architecture course in a public university in the Midwest of US for

several times. Based on the feedbacks from the students who have taken the introductory computer architecture course in previous semesters, we find that many of these students were concerned about the diverse topics covered in this course, and the relationship between different topics within this course, as well as the correlation between this course and other preliminary courses. Also, students often have difficulties in starting simulation and assembly programming projects at early stages of the course since they are lack of sufficient knowledge of the computer and the simulator framework as a system.

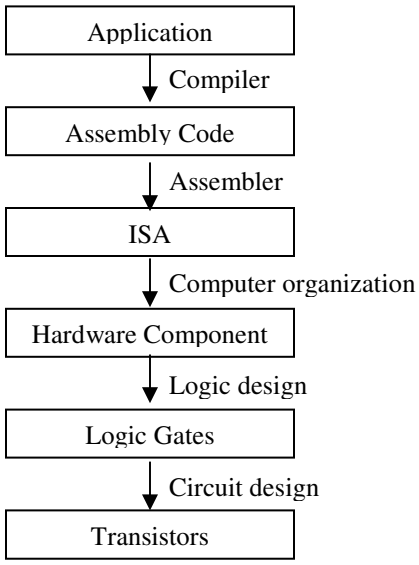

Figure 1. Abstraction levels of a computer system.

To help students overcome these difficulties and develop a better understanding of the computer architecture, I propose to put emphasis on the systematic view in teaching the introductory computer architecture courses, which has been implemented in the updated computer architecture course in our university. In the updated course, the instructor provides an overview picture of computer systems at the first class of the introductory computer architecture course. In addition to explaining that computer architecture (ISA, specifically) serves as the interface between the software and the underlying hardware, the instructor also emphasizes that both the hardware and software can be implemented in different levels of abstraction, as shown in Figure 1.

Precisely, as can be seen in Figure 1, the hardware components (e.g., ALU) can be implemented in logic level or transistor level, which helps students connect the hardware blocks in a microprocessor with what they have learned in logic design courses or VLSI design courses. Similarly, the software can be written by different levels of representation (e.g., high-level language or assembly language), which is finally translated into binary code before running on a computer. Such an explanation helps students understand the relation between the software development and computer architecture. It should be noted that the abovementioned structured computer organization [11] has been a widely-used concept. However, this paper introduces our

approach and experience in leveraging the levels of computer abstraction to improve students' learning outcome in the introductory computer architecture course.

Furthermore, the instructor introduces how a computer works in a high-level view before explaining any specific hardware component. More specifically, the instructor teaches students that the processor typically runs the following tasks: to fetch instructions from the memory (which is treated as a black box for now), to decode the instruction, to execute the instruction, to get the data from registers or memory, and finally to write back the results into registers or memory. We find that such an approach can help students make a smooth transition from the logic design and programming language courses to the study of computer architecture and organization. Also, such a systematic view about computers makes it much easier for students to perform assembly language programming and simulation project without worrying too much about the details of the "unknown" components, no matter it is a hardware component or a software component.

In addition to the overview at the first class, the instructor can reinforce this global picture before each hardware component is introduced in details. Specifically, the instructor can put the current component that is being taught into the framework of the whole computer system, and treat other components that have not been studied yet as black boxes. Such a global picture makes it simpler for students to understand where the current hardware block fits in the whole computer system, and how it interacts with other components. Moreover, this global view motivates students to explore the "unknown" hardware components so as to better understand how the whole system works.

2.1 Performance-centered systematic view of a computer

While power consumption and reliability have increasingly become critical design considerations for microprocessors, performance is still one of the most important design goals for most microprocessors, which is typically the main theme in the introductory computer architecture courses. In the updated course, the instructor uses the performance equations presented in the classical Patterson and Hennessy book [1] at different stages of the class to highlight the performance issue while erasing students' burden of possessing the indepth knowledge of various hardware components at early stages of the course.

More specifically, at the first overview class, we present the formula *CPU time* = *Instruction count* * *CPI* * *Clock cycle time* [1] to measure the performance of microprocessors. This formula clearly shows the impact of both hardware and software on the performance of the computer system. However, at this stage, we only require the students to understand that software and compiler can impact the *instruction count* and hardware implementation can affect the *CPI* and *clock cycle time*, which give them an overall picture on how to optimize both the software and hardware for achieving higher performance. A deeper understanding on how to reduce the instruction count by compiler optimizations can be learned in a compiler course or an advanced programming language course later on. After we introduce the design of a single-cycle processor and a multi-cycle processor, the students can understand how different hardware implementations can impact the *CPI* and the *clock cycle time*. Also, once students have learned the instruction

pipelining, they can understand more techniques to improve performance by reducing the *CPI* and the *clock cycle time* through pipelining. It should be noted that at this stage, the students still treat the memory and I/O devices as black boxes. After we introduce the cache memory, the students now understand that the *CPI* can be increased due to cache misses, thus hurting performance. Similarly, after we introduce the I/O devices, students begin to appreciate the fact that the performance can be limited by the I/O bandwidth as well, thus a balanced design is a necessity to enhance the overall performance of the computer system. To summarize, as the class progresses, we elaborate the performance formula step by step, which helps students gradually establish a global picture of the performance issues of microprocessors without overwhelming them with too much details.

2.2 Importance of systematic view in computer architecture education

With the rapid development of semiconductor and new technologies, and the innovations of hardware and software design, the content of computer architecture courses also needs to be updated to incorporate new architectures and parallel structures [3], as well as new technologies [5]. For instance, multithreaded and multi-core architectures have been increasingly adopted by major microprocessor companies, thus the fundamentals of multithreaded and multi-core architectures should be introduced to prepare students for the coming era of multithreaded and multi-core computing. In addition, as embedded processors are widely used in many systems, embedded processors are increasingly incorporated in the computer architecture courses, which are fundamentally different from the general-purpose microprocessors that have been taught in the traditional computer architecture courses for many years. With the increasing complexity in the computer architecture courses, we believe that the systematic view based approach can become even more important for future computer architecture courses to deal with the complexity.

3. System-oriented projects and assignments

In accordance with our philosophy of emphasizing the systematic view of computer systems, we design course projects, assignments to assist student in keeping the global picture in mind and understanding the relations between each hardware component and the computer system. While traditional projects and assignments mainly focus on each specific hardware component, we design system-oriented projects and homework by integrating different hardware components that have been covered in the course to reinforce the systematic view. We list some example projects and assignments as the following:

1. After we introduce the MIPS assembly language, pipelining and memory hierarchy, we ask the students to write an assembly program to implement the matrix multiplication with different data layouts (i.e., row major or column major) and evaluate the performance by using the SPIM simulator [2]. The difference in performance between different layouts of matrix clearly indicates the great impact of cache memory on the instruction pipelining and the overall execution cycles [6]. Thus, students are motivated

to optimize the data layout or transform the program to improve the cache performance, leading to higher system performance.

2. After we introduce the cache and virtual memory, we design an assignment by integrating both the virtual memory and cache. More specifically, we ask the students to calculate the average memory access time by taking the address translation time into account. Such an assignment can help students understand quantitatively how the TLB (Translation Look-aside Buffer) can speed up the translation of virtual addresses to physical addresses and its impact on the overall memory performance.

Moreover, in addition to two midterms, we use a comprehensive final exam to strengthen students' understanding of the computer as a system by integrating their knowledge of specific components. It should be noted that while at the beginning of this course, students have a high-level systematic view of the computer system by treating each component as a black box; by the end of this course, students are expected to have both a better understanding of the systematic view of the computer system and the indepth knowledge of the design of each individual component, which can be tested in the comprehensive final exam.

4. Emphasis of both hardware and software

The introductory computer architecture course at our institution traditionally focuses on hardware design and implementation. All the projects were based on the *Xilinx ISE* to create schematics and run logic simulations. The author has significantly updated this course by introducing the software design and its interaction with the underlying hardware since 2004. More specifically, the instructor added the assembly programming as an essential component to this course, which is centered on the MIPS processor, as described in the textbook [1]. We believe students' capability of developing assembly programs for the target processor (i.e. MIPS) will not only deepen their understanding of the ISA and performance, but also better prepare them for the increasing industry demands of capable engineers that possess proficient knowledge in both hardware and software domains. In this regard, we designed a set of assembly programming projects (in addition to the traditional *Xilinx* design projects), which are listed as the following:

1. Use MIPS assembly code to implement the following problem: given a number *N* (*N* is a positive integer), compute the sum *1* to *N*.

2. Use MIPS assembly code to implement a loop for calculating the factorial of *N*. For instance, given input *6*, it should return the output *720* (i.e., *6!*). Use a procedure call to calculate the factorial of *N*. Therefore, you only need to pass the parameter *N* to the procedure, and then to print out the value returned by this procedure, which should be *N!*.

3. Use pointers in MIPS assembly language to manage *dynamic memory structures* (i.e. *heap*). We can use *syscall 9* ($*a0*=amount, and the pointer address will be returned in $*v0*) to acquire memory space from the heap at runtime. A dynamic

string is dynamically allocated varying length string. Such strings end with a null character (*0*). Such strings are accessed via a pointer to the first character of the string. In this lab, you are required to develop a MIPS assembly program to support the following functions:

a. Inputs two strings from the keyboard, call them *strA* and *strB* respectively.
b. Appends *strB* to *strA*
c. Print *strA*

4. Use MIPS assembly language to implement the memory-mapped I/O by using polling. Specifically, your program should allow users to press the keyboard to input any letters (a-z) and your program should display those letters on the monitor. You are not allowed to use *syscall* in your program, which will make the lab trivial.

While the first two assembly programming projects familiarize students with the basic instructions, control flow and procedure calling convention of MIPS assembly language, the third project and the fourth project will provide students the experience to use assembly language to control the memory and I/O devices explicitly, which can help students to "see" and understand the interactions between software and hardware.

5. Performance of students

The author has taught the introductory computer architecture course in a research university for several semesters, with a very similar class size. In this paper, we compare students' performance of semester I (without the systematic view) and semester II (with the systematic view), which are shown in Table 1. In both semesters, while the questions in each test are different, the instructor has made efforts to ensure that they have same or very similar scope and level of difficulty. By emphasizing the systematic view of the computer system in semester II, the author found that the students became more confident and interested in the course projects and assignments, and more active in the class in general. Compared with the previous semester without employing such an approach, students' performance in semester II, in terms of the average score in the midterm and final exams, has been adequately improved, indicating the effectiveness of the systematic teaching approach in this course.

Table 1. Averaged score of two introductory computer architecture classes with and without the systematic view based approach.

	Average Score of the Class (full score:100)	
Exams	Semester I (w/o systematic view)	Semester II (with systematic view)
Midterm I	76.3	82.5
Midterm II	75.8	81.6
Final	65.6	85.5

6. Comparison to other pedagogical approaches

There have been several related proposes [7, 8, 9, 10] to teach computer architecture courses in the literature. Patt and Patel [7] proposed to teach low-level hardware component with C programming. Bryant and O'Hallaron [8] advocated an approach to teaching architectural concepts from the software programmers' perspective. These two approaches [7, 8] are complementary to the systematic approach presented in this paper, which can be used to enhance the teaching of the hardware components, and the software and hardware interactions.

Recently, Saltzer and Kaashoek [9] introduced a pedagogical approach to teaching a computer system engineering course at MIT. Ramachandran and Leahy [10] presented an integrated approach to teaching computer systems architecture at GIT. Both these two pedagogical approaches [9, 10] are similar to ours in spirit, although all these proposals were developed independently. Also, Saltzer and Kaashoek's approach [9] focused on the general principles and abstraction of engineering computer systems, regardless of a computer or an operating system, a client/server application, a database application, or a fault tolerant disk cluster. In contrast, our approach specifically concentrated on the introductory computer architecture course, and we have presented detailed information on how to organize the course to emphasize the systematic view, as well as sample labs and assignments. By comparison to Ramachandran and Leahy's approach to teaching computer architecture and operating system in an integrated manner [10], this paper focuses on providing students' the systematic view to improve their learning outcome of studying computer architecture.

7. Concluding Remarks

Traditionally, the introductory computer architecture course focuses on the in-depth study of various hardware components of computer systems, whose complexity often impedes students' learning process if not balanced with a global view of the system. This paper introduces the emphasis of the systematic view of computers in the introductory computer architecture course to help students overcome this learning obstacle. Such an approach is especially useful as the content of computer architecture course is continuously expanded and becomes more complex, with the relentless advancement of technology and computer architectural innovations. The introductory computer architecture course needs to provide a balance between the high-level understanding and the component-based, low-level implementation of the computer systems. It is expected that the emphasis on systematic view can help students improve the learning outcome of the introductory computer architecture course.

References

[1] David A. Patterson, John L. Hennessy. Computer organization and design, the hardware/software interface. Morgan Kaufmann Publishers, 2005.
[2] Homepage of SPIM simulator. http://www.cs.wisc.edu/~larus/spim.html
[3] Sally L. Wood, Chris Dick. Concepts of parallelism in an introductory computer architecture course with FPGA laboratories. In Proc. of the ASEE/IEEE Frontiers in Education Conference, 2004.

[4] N. Calazans, F. G. Moraes and C. Marcon. Teaching computer organization and architecture with hands-on experience. In Proc. of the 32nd ASEE/IEEE Frontiers in Education Conference, November 2002.
[5] M. T. Niemier and P. M. Kogge. Teaching students computer architecture for new, nanotechnologies. In Proc. of the Workshop of Computer Architecture Education, 2002.
[6] L. Pascual, A. Torrenti, J. Sahuquillo and J. Flich. Understanding cache hierarchy interactions with a program-driven simulator. IN Proc. of the Workshop on Computer Architecture Education, June 2007.
[7] Y. N. Patt and S. J. Patel. Introduction to computing sytems: from bits & gates to C & beyond. McGraw-Hill.
[8] R. E. Bryant and D. O'Hallaron. Computer systems: a programmer's perspective. Prentice Hall, 2003.
[9] J. Saltzer and F. Kaashoek. A systems approach to teaching computer systems. In Proc. of WCAE, 2006.
[10] U. Ramachandran and W. D. Leahy Jr. An integrated approach to teaching computer systems architecture. In Proc. of WCET, 2007.
[11] A. Tanenbaum. Structured computer organization, fifth edition. Prentice Hall, 2005.

Biographical Information:

WEI ZHANG: Prof. Wei Zhang received the Ph.D. degree in computer science and engineering from the Pennsylvania State University in 2003. He joined the ECE Department at Southern Illinois University Carbondale as an assistant professor in August 2003 and has become an associate professor since July 1, 2007. His research interests are in embedded computing systems, computer architecture and compiler.