The Architecture Curriculum at UC-Davis

Matthew Farrens

Department of Computer Science
1 Shields Avenue
University of California, Davis
Davis, CA 95616-8562
(farrens@cs.ucdavis.edu, http://arch.cs.ucdavis.edu/~farrens)

Abstract

In this paper an overview of the architecture curriculum at UC-Davis will be presented, with an emphasis on the classes offered through the Computer Science Department. Each particular class will be described briefly, and then some personal opinions about one class in particular and experiences with teaching in general will be presented.

1. Architecture in the Computer Science Department

The Computer Science Department at the University of California, Davis supports majors in both Computer Science (through the College of Letters and Sciences) and in Computer Science and Engineering (through the College of Engineering). Our core architecture curriculum consists of 6 classes - 3 at the undergraduate level and 3 at the graduate level. Table 1 lists the classes, each of which will be described in more detail in the following subsections.

Table 1. Architecture Classes

Type	Name	Title
UGrad	**ECS50**	*Computer Organization and Machine Dependent Programming*
UGrad	**154A**	*Computer Architecture*
UGrad	**154B**	*Computer Architecture*
Grad	**ECS250A**	*Advanced Computer Architecture*
Grad	**ECS250B**	*High-performance Uniprocessing*
Grad	**ECS250C**	*Parallel Processing*

Davis is on the quarter system, with each quarter being 10 weeks long, so there is a total of 30 weeks of architecture instruction at each level.

1.1. ECS50

This is the first architecture class students take, and uses Assembly Language as a tool to convey basic concepts in computer architecture. The students write programs on both CISC and RISC architectures, in order to ensure they are exposed to both types. Basic architectural concepts such as addressing modes, memory layout, instruction set design, etc. are covered in this course. Interrupts are also covered here, since writing interrupt routines requires an understanding of the architecture and assembly language of a particular machine. This is designed to be a very challenging class, with homework and/or programming assignments due 9 out of the 10 weeks.

1.2. ECS154A

This is something of a transitional class, since students who are majoring in Computer Science (CS) have slightly different degree requirements than those majoring in Computer Science and Engineering (CSE). In particular, the CS students are not required to take any digital design classes, while the CSE students are. Thus, the major topics covered in 154A are digital design, I/O (busses, etc.), and an introduction to the memory hierarchy (caches and virtual memory). The digital design portion of the course can be a bit tricky - the goal is to give the CS students enough information to allow them to understand and work with the material in ECS154B, without wasting the time of the CSE majors who have either already had or will soon take a class exclusively about digital design.

1.3. ECS154B

In 154B, more advanced architecture topics are covered - hardwired and microprogrammed CPU design, an introduction to pipelining and multiprocessors, advanced topics in memory hierarchies, uniprocessor performance analysis under varying program mixes, etc. The digital design tools introduced in 154A are used here to actually design different parts of a real CPU.

1.4. ECS250A

In our program, all graduate students are required to take this class. Thus, it covers a fairly wide range of topics and has to be accessible to students with a fairly wide range of backgrounds. Topics covered in this class include the fundamentals of computer design, instruction set principles, pipelining, an introduction to instruction level parallelism, memory-hierarchy design, and some parallel processing. This class includes a project, which is often difficult to do in a 10-week course (since the more interesting material isn't covered until the second half of the quarter).

1.5. ECS250B

This class focuses on the architecture and design of high performance uniprocessors. In this class the students are exposed to the pioneering work done on supercomputers like the IBM 360/91, the CDC 6600, and the CRAY. In addition to studying the classics, we look at instruction level parallelism in detail, and also spend time studying the memory system bottlenecks and how they might be circumvented. Students do projects and present results to the class as a whole, to practice presentation skills.

I try to emphasize to students the reasons for studying the old machines - many (if not most) of the problems confronting designers today were faced by designers of these machines as well (processor/memory speed imbalance, pipelining hazards, etc.). I try to convince the students that they need to study and be familiar with these old, dilapidated ideas because as technology changes, many of the old ideas are rediscovered and become relevant again. For example, a technique used 30 years ago may not translate directly into the modern era, but two old techniques used together in a novel way might.

1.6. ECS250C

This class emphasizes how to do research in and on parallel architectures, from special-purpose machines to commodity servers. The class is systems-oriented, covering a broad range of topics including VLSI, architectures, operating systems, compilers, and applications. The course emphasizes critical reading of current research papers and the final project focuses on written and oral presentation of original results.

2. Architecture in the Electrical and Computer Engineering Department

The ECE program differs substantially from the one in Computer Science. It has more of an emphasis on hardware, and features a two class sequence (*Computer Structure and Assembly Language* and *Introduction to Computer Architecture*), augmented by several specialty classes (*Microcomputer-Based System Design*, *Digital Systems*, etc.) Since I am a member of the Computer Science Department and not the Electrical and Computer Engineering Department, I will not comment further about their program.

3. The Importance of Fundamentals

I feel very strongly that in many respects the first architecture class the students are exposed to (ECS50) is the most important class of the entire sequence. If handled right it can set the tone not only for the rest of their architecture classes, but also for many of their other classes as well.

When teaching this class, I have three goals for the students:

1. To gain a solid understanding of the fundamentals of Computer Architecture,
2. To learn how much alike all Von Neumann machines are, and
3. To be able to program any machine with a minimum of effort (in essence, to be able to read a manual).

I explain to the students that even though people rarely get jobs these days writing programs in assembly language, what they will learn in this class will still be of use to them in the future. The analogy I generally use is that you don't have to know how a car works to be able to drive it, but the more you **do** know about a car the better driver you become. Knowing what a car does and how it works helps you get better gas mileage, makes it last longer, helps you make smart decisions when driving (like *why* to avoid the potholes, why you should change your oil regularly, how to avoid accidents, why not to ride the brakes, etc.). Knowledge is power, and my goal is to fill their knowledge gas tanks.

I believe it is extremely important to introduce and emphasize concepts and fundamentals. Understanding the fundamentals is key to a good solid working knowledge of architecture in general. Personally, I am opposed to using existing architectures to teach these fundamentals, because all commercial products have aspects that exist purely because of manufacturing restrictions in place at the time the processor was designed. Unfortunately, students who are seeing this material for the first time are frequently unable to differentiate between fundamental concepts and machine-specific restrictions and idiosyncrasies.

Because of this, I use a very clean generic machine as a teaching tool. This allows me to convey what is important and universal about all Von Neumann machines. Once this knowledge has been imparted, I can move on to existing architectures and show that they contain all the basics elements presented using the generic machine, as well as their own individual warts and bumps. As technology constantly leads to changes in implementations and architectures, a solid grounding in why certain features exist is crucial.

I try to focus on clarity of presentation and clarity of concepts. I want the students to build a simple mental model of a machine, which will be refined and expanded throughout their education. Using a generic teaching architecture provides a vehicle to convey *concepts* to students clearly and unambiguously. By taking this machine and morphing it into existing architectures, I am able to explain how the concepts remain the same - only the semantics/organization changes. My goal is not to teach the student how to program commercial product X, but rather to enable the student to (with a little work) program **any** commercial product. I think this is particularly important at UC-Davis, given that we are a University. Trade schools exist for those who only want to learn how to write programs for one particular architecture.

I do not use a 32-bit load/store architecture as the teaching architecture because its sheer size can be very intimidating to students not well versed in the ins and outs of computer architecture. Such an architecture also has several features (again due to implementational considerations) that the naive student cannot distinguish from necessary features. Using a load/store machine exclusively, for example, prevents students from learning about and experiencing the various addressing modes that exist in many architectures. I use a nice simple generic architecture that allows me to convey the underlying principles directly without having them obscured by particular implementations.

I try to have an explanation for everything that I present. As a student, I was never satisfied with the answer "Trust me for now, it will become clear later on". My goal is to ask the students to "trust me" as infrequently as possible. I begin the class by pointing out how fascinating computers actually are - they are nothing more than millions of on/off switches, arranged in groups of various sizes, with somewhat arbitrary meanings assigned to these groups. The magic is not in the silicon, but in the way humans have assigned meanings to these groups.

We then talk about what is necessary to do a computation (data, instructions), the subgroups within these (integers, characters, floating point), and how many switches (bits) should be allocated to a given group and why. After we get through the different possible number systems, character representations and ways to encode instructions, we then talk about Memory (a linear array of storage cells) and introduce a 3-operand instruction set that uses only direct addressing (the address of the storage cell is right there in the instruction). Doing this motivates the need for some local Memory (storage cells that are close and have their own special names, called *registers*). It also provides an excellent mechanism for getting across multiple addressing modes (immediate, direct, register, etc.).

This approach helps the instructor answer the standard student question, "why did they do it this way?" If the job of explaining the fundamentals is done sufficiently, the student will be able to answer this question for him/herself. Or, alternatively, the professor can ask the students this question, and actually expect to get a meaningful answer!

4. My General Philosophy on Teaching

Since you are reading this, you are most likely interested in improving as an educator. In this section I will share some of my experiences and opinions on how to become a better teacher.

4.1. Find your own voice

This one takes a while, but everyone needs to find a style that works best for them. If you try to teach "just like Bob", chances are you will not be happy with the results (nor will your students). We happen to have 3 faculty members in our department who have won campus-wide teaching awards, and it took me a long time to stop trying to be just like them. I eventually realized that you have to do what works *for you*.

What matters most is enthusiasm for the topic. Whatever style works best for you, do that thing. But never underestimate importance of enthusiasm - if you don't care about the subject, why should the students?

4.2. Talk to other Educators

Assuming you are in this business to figure out how best to convey information to our students, make sure you take the time to talk about teaching with other educators. Even after you have established your "voice", you can always learn new things from others in the same business. I go out of my way to spend time talking to other professors about teaching techniques, and I almost invariably leave the discussions with some new ideas on how to approach or present a topic. Even when you are talking to someone who has a completely different opinion of how to do things, the exercise of explaining and defending your position can often reveal weaknesses that need improving.

4.3. Get students involved!

It's their money. Tell them that. Try to make them participate as much as possible in their own education. Personally, I don't call on people because I hated that when I was a student, but I do ask questions of the class as a whole and stop until I get a response. Sometimes there are long, uncomfortable silences, but eventually someone will gather the gumption to throw out a response.

Also, Make sure to explain to the students why you are doing certain things. For example, I give closed-book exams, but let them bring 1 page of hand-written notes. I discovered that when I explained my reasons for doing this - that I don't feel I can write as fair an exam when it is open book, and that the process of writing down on a sheet of paper the important stuff helps them organize and remember the material - the students were much more comfortable with the situation.

4.4. Identify Your Audience

Are you teaching to the top third? To everybody? To the middle? Deciding this can make a big difference in your approach. After attending this workshop a couple of years ago, and learning about the importance of having students working in groups, I realized that I had been avoiding groups because I was afraid students would be able to ride the coattails of their partners. I was more worried about students getting away with stuff than I was interested in maximizing the opportunities of the students who really wanted to learn. I have since modified my approach.

5. Tricks of the trade

In this section I will list a few gotchas I have learned over the years. Most people probably know all these things already, but I didn't. So in case you are as thick as I was ...

5.1. Exams

- Avoid redundant questions on exams. Make a list of the topics you want to test the students on, and then select questions from each topic. Otherwise, you can unintentionally have one topic carry a disproportionate amount of weight.
- Open-book tests are tricky - I find it very difficult to write a fair open-book exam. If the book is open, I tend to ask less general and more specific nit-picky kinds of questions. Personally, I don't want to know how well students can read a book, I want to know how well they know the material.

  What works best for me is a closed-book exam, with the students able to bring in one (or more) sheets of hand-written notes. I do this because I think the process of creating this sheet is more important than the end product - the student is forced to go through the material and identify the important topics, and the act of *writing* down this information helps drive home the material. (However, make sure you insist on a single sheet of **hand-written** notes - I discovered the hard way that enterprising students were photo-reducing entire sections of the book, which clearly defeats the entire purpose of the exercise.)
- Short answer questions are the best way to find out what a student knows, but be careful - on a timed test, long word problems are unfair to slow readers and to non-native english speakers. Such questions are also much harder to grade. If you have a question that you really think is important (like asking the students to work with a made-up architecture to see how well they understand certain topics) you can post the set-up part of the question the evening before, so everyone has time to read and digest the new information.
- I always give lots of partial credit. I have even given points for answers that were erased (not full credit, but they did have it right at one point!). Doing so helps convince the students that you really want them to succeed.
- If you decide to give partial credit, be very careful with questions that build upon previous answers. If they happened to get the first step wrong, you may have to use their incorrect answer to see if they got the following steps correct. This can lead to some *very* long grading sessions.
- Consider selling partial answers. I tell my students that if they get stuck on some part of a problem, particularly if it is a long, high-point problem, I am willing to "sell" them hints and/or partial answers for some number of points. This can often help a student get "over the hump" on a problem and prevent them from getting completely shut out.
- My graduate student has used this one - make the exam worth a million points. Then, when a student comes in to complain about not getting enough credit on a problem, you can give them a couple of thousand points and they leave happy!
- If you experiment a lot with various test formats, questions, etc. you almost certainly will need to grade on a curve. It is not fair to the students to penalize them for a test question that didn't work out the way you expected.

5.2. Other Tidbits

- The question of what late policy to use is always a tricky one. I use the following: Each student gets a total of seven late days (one week) during the entire quarter. They are responsible for managing these as they see fit. If they want to use them all up on the first homework, then when Grandma dies in week six they have a tough decision to make (because they are not getting any extra days). It is not difficult to keep track of this number, and this policy seems to work well. The problem comes when students work in teams - the solution to this problem is left to the reader.
- I learned about the value of student teams at the Teaching workshop a few years ago. One problem arises - if you are teaching a lower-division course, you need to decide how to help students find partners. It tends not to be much of a problem once the students have been on campus a while, but entering students can find hooking up with somebody very daunting.
- We grade all lab assignments interactively in the introductory class. The students schedule a time with the TA, and then sit down and the TA goes through the assignment asking questions about how and why the student did various things. This seems to be a very effective approach - the TA doesn't have to try and figure out what was going on with somebody else's program when it doesn't work, and it helps identify the students who have tried to pass off last quarters assignment as their own.
- One way to cut down on cheating is to use a seating chart. This way, the students may not know who they are sitting next to and therefore may be reluctant to copy off of them. However, seating charts add a level of complication that isn't always necessary. A simpler way is to have the students leave the tests on the desk when they finish. If you then pick up the tests in order and keep them that way, it is possible to

tell if matching answers were on neighboring tests. This has saved me on several occasions from sending exams over to our Student Judicial Affairs office. (Two tests had identical incorrect answers, for example, but the students were sitting in opposite corners of the room.)

- A colleague of mine rolls a pair of dice, and the student matching that number is called to the front of the room to give a 5-minute overview of the material presented during the previous class. This can be a very effective way of ensuring students are keeping up.
- Should you use overheads or the board? This is a difficult question. If you use overheads, you have to worry about going to fast. In addition, I am a firm believer in the value of writing things down (helps drive the material further into the brain). On the other hand, if students are spending all their time writing stuff down, they often are not able to follow the lecture material as closely. Personally I use the board and only use overheads for things like sample programs, but there are strong arguments on both sides of this one.
- Try innovative approaches - in 154B, I have the students design a machine from scratch interactively. After going over all the necessary background stuff, I tell them that I am now nothing but a scribe and tour guide and they are to propose an architecture that we will implement. Over the course of several lectures they select the wordsize, the number of registers, the instructions, instruction formats, etc. I point out the potential problems with various choices, but do not tell them they can or cannot do anything. I think it is valuable for the students, who get to see how decisions made at one point in the design cycle come back to haunt them later on. It does make life interesting for the professor, since it is not possible to prepare handouts in advance. However, overall I think the approach is worth all the extra work.
- On the other hand, engineers often don't "go with the flow" well - sometimes they don't take well to innovative (half-baked) ideas. Many engineers don't like discord and ambiguity, so you may have to steel yourself for some complaints on your teaching reviews.
- Cell-phones can be handy - I was sitting in my office one afternoon and got a call from my colleague (Fred Chong) who was giving a midterm. Turns out he had not printed enough exams, so I cranked out a few more for him, rushed them over, and the problem was quickly resolved. Slick!
- If you do **not** have a cell-phone (like me), the car can be a great place to read and grade. Nice soft chair, good sound system, no phones, great view...