Towards A Modern Computer Architecture Curriculum

Alan Clements, School of Computing, University of Teesside,
Borough Road, Middlesbrough TS1 3BA, UK.
Alex Shvartsman, Computer Science and Engineering, 191 Auditorium Road, U-155,
University of Connecticut, Storrs, CT 06268, USA.
Willis K King Computer Science, University of Houston,
4800 Calhoun Blvd., Houston, TX 77204-3475, USA.
Chao Lu Computer and Information Sciences, Towson University, 8000 York Road,
Towson, MD 21252-0001, USA.
Denis Dupont Universite d’Evry, Cours Monseigneur Romero, 91000 Evry, France.

Abstract – An EU-USA grant has been awarded to a consortium of six universities to exchange students across the Atlantic and to facilitate the mutual accreditation of degrees between US and EU members of the consortium. One of the goals is to develop courses in three areas that could be taught and mutually recognized by all partners. “Computer Architecture” is one of the areas selected for investigated by the consortium. This subject is widely taught in many universities, although both the level and the specific contents vary from institution to institution. The consortium has looked at the computer architecture curriculum suggested by the joint ACM and IEEE Computer Society CC-91 report, and typical courses in universities in both Europe and the USA.

Fundamental developments in computer architecture have made it necessary to restructure the architecture curriculum. The proposed sequence consists of three courses and associated laboratories. A first course, CA1, provides a breadth-before-depth approach to computer architecture and covers a wide range of topics from number bases to the computer systems. Students in both CS and other computing programs may take this course, because it provides a complete picture of the computer system. CA1 will introduce elementary digital circuit design but will not concentrate on logic design. However, an optional laboratory component will enable some universities to cover digital systems in greater detail.

The second course, CA2, provides more depth and will be taken by all computer science and computer-engineering students. In particular, one or more processor architectures will be studied in detail. Because of the synergy between architecture and operating systems, CA2 will include architectural support for operating systems. CA2 will also provide an introduction to multiprocessors and their interconnection networks.

The third course, CA3, provides an opportunity to introduce the elements of modern computer architecture that have been responsible for dramatically enhancing the performance of computer systems. In particular, CA3 will concentrate on instruction level parallelism, speculative computing, and support for multimedia applications. Issues in multiprocessing and distributed computing such as cache coherency, processor synchronization, multiprocessors, and network computing will be addressed.

These three courses provide both breadth and depth in computer architecture. CA1 provides breadth. CA2 provides depth in computer architecture and CA3 examines modern issues in high-performance processing.

Background

The governments of the European Union and the USA have set up a program to promote undergraduate student mobility in both directions. A consortium of six universities in the USA, France, Germany, and the UK has been awarded a grant to examine the computer science curricula in the four countries of the consortium in order to stimulate collaboration and standardization of degree programs in computer science. Part of the project will involve the exchange of up to 30 students. The ultimate goal of the project is to establish the mutual recognition of accredited degrees in computer science between the EU and the US.

Undergraduate programs in all countries involved in this project appear to be in substantial agreement on the core materials that should be included in a degree curriculum. The members of the consortium have decided to develop three components of the CS curriculum to facilitate the exchange of students and simplify accreditation. Reaching better understanding of course contents and degree requirements can lead to the mutual recognition of academic degrees which will, in time, facilitate the practice of the profession across national borders. This is essential in the interest of promoting international trade and commerce.

Computer architecture was selected as a suitable candidate for joint development for three reasons. First, all universities teaching computer science provide one or more courses in computer architecture. Second, several members of the consortium have had previous experience in developing computer architecture courses. Finally, computer architecture is a key discipline that underpins computer science itself. Although computer architecture is sometimes
defined as the assembly language programmer's view of the computer, we take it to include number representation, logic design, and computer and system organization.

Re-evaluating the Computer Architecture Curriculum

At first sight, the task of creating a new computer architecture should be easy. A computer architecture course at one university is broadly similar to those at other universities. The only real differences lie in the emphasis given to individual topics. Moreover, in 1991 the ACM and IEEE Computer Society produced a suggested curriculum for computer architecture [1]. Why, therefore, was it necessary to invest substantial effort in devising a new curriculum in computer architecture?

In the eight years since the CC-91 guidelines were written, major changes have occurred in computer science. The discipline has expanded immensely both in terms of breadth and depth. Topics that existed in 1991 now require a greater depth of understanding because of the latest developments, and new fields (e.g., object-oriented programming, network and parallel computing, and visualization) have emerged and matured to the point where they need to be reflected in a modern curriculum.

Such an increase in the body of knowledge making up computer science has put immense pressure on academics to justify the inclusion of their particular discipline in the computer science curriculum. Nowhere is this more evident than in the case of computer architecture. Some CS educators have even expressed opinions that undergraduate study of computer architecture has become less relevant and that it should be minimized within the curriculum to make room for more software-oriented subjects.

In the past decade immense strides have been made in the field of computer architecture. However, much of this progress is invisible to the non-specialist in architecture. Many academics teaching computer architecture will probably have been engaged in a conversation where a software-oriented colleague suggests to them that computer architecture is now irrelevant to the discipline. They argue that most students will never have to program in assembly language and they will certainly never have to build digital circuits with gates. Before we introduce the proposed courses, it is worthwhile indicating why computer architecture should remain an important component of the computer science curriculum.

The Role of Computer Architecture in the Curriculum

The computer lies at the heart of computer science--without the computer, computer science would be a branch of theoretical mathematics. Consequently, it is reasonable to teach university students how computers are organized. A student should not just regard the computer as a black box that executes programs as if by magic. Moreover, recent advances in computer architecture are every bit as exciting as advances in physics and biology.

An understanding of computer architecture has important implications for the practitioner of computer science. Suppose a graduate enters the industry and is asked to select the most cost-effective computer for use throughout a large organization. Understanding how the various elements of a computer contribute to its overall performance is vital. For example, it is important to have an educated answer to the question of whether it is better to spend, say, $50 on doubling the size of the cache, or is it better to spend $100 to double the clock speed?

Computer architecture cannot entirely be divorced from software. A student constructing a web page on a PC page may be blissfully unaware of the underlying architecture. However, the vast majority of computers are not found in personal computers or workstations. Most computers are in embedded applications where architectural considerations are very important. Those designing multiprocessor systems, real-time systems, or systems where synchronization is important have to understand fundamental architectural concepts and limitations. Someone developing an automobile electronic ignition system may write their code in C, but may have to debug the system using a logic analyzer that displays the relationship between interrupt requests from engine sensors and the machine-level code.

A knowledge of computer architecture can help computer scientists to perform their jobs better. Consider the following analogy. A professional driver who understands engines and transmission systems can get better performance and reliability that someone who lacks that knowledge.

Another reason for teaching computer architecture is that it incorporates a wealth of important concepts. This point is probably least appreciated by computer scientists who took a course in architecture some time ago and did little more than learn about byte, gates and assembly language. Computer architecture can be used as a tool to support other areas of computer science via recurring themes. For example, studying how the computer provides architectural support for high-level languages reinforces the teaching of pointers in C. Teaching bus design and arbitration introduces important topics such as “fairness” versus “priority”. Similarly, understanding cache coherence protocols for multiprocessors allow developers to design better cache management approaches for web browsers.

Changes in Computer Architecture

Computer scientists in the 1980s began to reevaluate architectural principles and were responsible for the development of a new generation of computers that exploited instruction-level parallelism, the so-called “reduced instruction set," or RISC machines. Executing instructions in parallel led to the development of techniques
that could predict the outcome of branches before they were executed in order to prefetch instructions from the branch target address. Pipelined and superscalar processor design, even for RISC architectures, have instruction sets where individual instructions cannot be interpreted in isolation from the surrounding instruction sequence. New memory models have been proposed and implemented. These models are difficult to reason about and require new formalisms to specify their behavior. Similarly, high degrees of instruction level parallelism have lead to concepts such as speculative execution in which the processor guesses the value of a result (for use as a source operand) before it has actually been calculated. Additionally, designing processors that can be routinely deployed in multiprocessor configurations led to the development of new efficient processor synchronization primitives.

Any new curriculum in computer architecture should reflect such state-of-the-art developments.

The Curriculum

In this section we present the new computer architecture curriculum. We begin by giving the rationale that led to the development of the sequence of three courses that constitute the curriculum.

Philosophy of the Courses

Because computer science has grown so much over the last decade, no single degree program can do justice to all its aspects. This expansion has led to a fragmentation of computer science and the generation of new degrees. Complementing traditional Computer Science, and the established degrees in Computer Science and Engineering, and Computer Engineering, some universities now offer degrees in Information Technology, Software Engineering, Visualization, and Multimedia. Some of the new degrees place less emphasis on traditional computing (e.g., data structures, languages and compilers, and AI) and more on the applications of computers. In particular, there is a tendency not to include computer architecture in some of the newer curricula.

Given a situation, in which some wish to abandon computer architecture whereas others wish to expand it because of the recent developments in architecture, the curriculum designers find themselves in a dilemma – do we provide more, less, or continue as before? The proposed architecture courses attempt to deal with these issues by taking a breadth-before-depth approach in the first architecture course and then specializing and pursuing comprehensive coverage in later courses. We have formulated a three-course sequence called CA1, CA2, and CA3. In this sequence, CA1 aims to provide foundations in a way that makes it suitable for most programs in computing, whereas CA2 and CA3 may be optional for some programs. CA2 should be required by all computer science and computer engineering programs and CA3 may be required in programs whose goal is to incorporate deeper coverage of computer architecture, computer system design, and network and parallel computing. Each lecture course can have an associated laboratory that provides additional details for selected topics. Laboratories are included in programs based on their specific requirements.

An important feature of these courses is the support they provide for other areas of computer science, notably languages and compilers, operating systems, parallel systems and networks. For example, the electrical link layer and data link layers of data transmission systems can be covered in a computer architecture course because they display a closer affinity to architecture than to the more software-oriented network courses. Understanding modern instruction set architectures is indispensable in a comprehensive study of code generation in compiler courses and the study of process/processor synchronization issues in operating system courses.

The First Course: A Breadth before Depth Approach

The first course, CA1, takes a “breadth before depth” approach by combining elements of the traditional digital logic course with the introductory level architecture course. Ideally, CA1 should be taken after an introduction to computer science. The goal of this course is to ensure that all students get an appreciation of the computer system from the gate level to the system level. Of course, the effect of combining two courses is to reduce the depth at which the material can be taught. The lack of depth can be made good by means of an optional laboratory course together with CA2 and CA3. The following syllabus defines CA1 and can be compared with the CC-91 syllabus in the appendix, which also provides the next level of detail for common topics.

CA1 The first course in computer architecture

1.0 Introduction to Computer Systems
   Fundamental components (CPU memory, I/O)

1.1 Number representation
   Binary, hexadecimal, decimal
   Base conversion
   Negative numbers and 2s complement arithmetic
   Addition/subtraction
   Fixed and floating-point arithmetic

1.2 Digital logic
   Gates and simple circuits
   Flip-flops, counters, registers, ALU
   Tri-state devices and buses

1.3 Instruction set architectures
   Data and address paths
   Structure of instructions
   Addressing modes
By the end of CA1 students should have a good understanding of the computer system. They will be able to appreciate and use binary arithmetic but will not, for example, cover high-speed multiplication techniques or the more sophisticated logic simplification techniques.

CA1 will be especially suitable for students taking, for example, degrees in Visualization or Information Science (often called IT Informatics in Europe) or because CA1 provides a complete picture of the computer system and does not require a further course. At the same time, it introduces some of the concepts that naturally lead to more advanced courses (e.g., parallel processing).

In order to cater to students on degree courses that go further into digital logic and assembly language programming, this course can be supplemented by an optional laboratory based course.

Having taken this introductory course, students on computer science or computer engineering degrees are in a position to take CA2 that goes more deeply into computer architecture.

The Second Course: Modern Processors and Hardware-Software Interface

The second course in computer architecture, CA2, performs three important functions. First, it provides a conventional course in computer architecture that takes an in-depth look at a modern processor. Second, it covers topics that illustrate the hardware-software interface (e.g., architectural support for operating systems). Third, it provides a starting point for the next level course, CA3.

CA2 The second-level course in computer architecture

1.0 Introduction to computer architecture
   - A case study
   - Instruction set architecture (ISA)
   - Pipelining
   - Introduction to superscalar design
   - Instruction parallelism
   - Comparative architectures
   - Low-level support for high level languages
   - Cache memory

2.2 Architectural support for operating systems
   - Memory management
   - Multitasking and interrupts
   - Synchronization primitives
   - Protection mechanisms

2.3 The I/O subsystem
   - Bus structures
   - Storage technology
   - Multimedia systems

2.4 Multiprocessing and networks
   - Fault tolerance
   - Introduction to networks
   - Parallel processing concepts, SIMD/MIMD

As in the case of CA1, this course may include an optional laboratory component for those universities that wish to place a stronger emphasis on computer architecture. Such a laboratory course might explore the design of computer architectures via simulation or hands-on exploration using a specific processor. Equally, the laboratory might, for example, use hardware design language, e.g., VHDL or Verilog, to support design of processor components and parallel computing structures.

The Third Course: Performance, Parallelism and Networks

The third-level course, CA3, will be taught in universities wishing to provide an in-depth treatment of issues in modern computer architecture. In particular, this course emphasizes the ways in which the performance of modern processors and systems is accelerated. Following the introductory material in CA2, the third course explores in more depth parallel processing and network computing. This course will also include advanced in audio-visual techniques used in multimedia systems (e.g., architectural support for multimedia operations and the operation of AV disks).

The later part of this course covers the more traditional material found in advanced computer architecture courses such as parallel processing.

CA3 The third-level course in computer architecture

3.1 Enhancing performance
   - Branch prediction
   - Superscalar architectures
   - VLIW architectures
   - Prefetching
   - Speculative execution
   - Multithreading
   - AV support
   - RAID architectures
   - Scalability

3.2 Parallel processing and tightly coupled systems
   - Systolic architectures
   - Shared memory systems
Interconnection networks
Hypercube, butterfly, shuffle-exchange, crossbar topologies
Cache coherence protocols
Memory models and memory consistency

3.3 Networks and distributed systems
Introduction to LANs and WANs
Layered protocol design, ISO/OSI, IEEE 802
Impact of architectural issues on distributed algorithms
Network computing
Distributed multimedia systems

This course may include an optional laboratory component where programs require it. Such a laboratory course continues to explore the design of computer architectures via simulation or hands-on exploration. The laboratory might use a hardware design language to support design of complex processors and systolic systems. Design of, and experimentation with LANs, e.g., Ethernet, can also be readily included in the laboratory.

The Role of History in Computer Architecture

Students should have some knowledge of history of computer architecture – not least because it is instructive to see how historic trends, economic pressures and political pressures have influenced the design of computers. Although the three proposed courses do not explicitly include historical themes, it is suggested that those teaching courses in computer architecture should put topics in their historic context wherever possible.

Course Sequencing

Figure 1 suggests how these modules can be included in a typical four-year, semester-based degree course. This table also suggests the timing of the introductory CS1 and CS2 courses in programming, data structures and algorithms. Typically, CS1, CS2, CA1 may form the normal three-course sequence included in the first three semesters of most computer science programs. We also suggest the location of the courses on operating systems and compilers. However, the actual sequencing of actual programs will depend on the nature of the institution and its objectives.

The associated laboratory components of the proposed curriculum are normally positioned within the timeframe of the respective lecture courses. Some programs may choose to detach the laboratory courses and create full courses, i.e., 3 or 4 credit courses, which focus on laboratory work. In these cases the laboratories can be taken concurrently with associated lecture courses or subsequently to the lecture courses.

Conclusions

The proposed courses in computer architecture have been designed to enable all students in programs in computing to get an overview of computer architecture and computer system, while providing the necessary depth for students who wish to specialize in computer architecture and computer engineering. The courses form a sequence so that one course naturally leads on the next. Each of the three courses can be supplemented by optional laboratory-based courses.

One significant feature of the courses is an emphasis on modern themes in computer architecture, for example we include architectural support for today’s multimedia systems. Another important feature is the support for related areas of computer science including languages and compilers, operating systems, and computer networks.

Acknowledgments

The authors thank the EU-US consortium for supporting this work. We also thank Michel Israel for his constructive participation in many of our discussions.

Reference


Appendix The CC-91 Model Architecture Curriculum

The knowledge units in the common requirements for the subject area of Architecture emphasize: digital logic, digital systems, machine level representation of data, assembly level machine organization, memory system organization and architecture, interfacing and communication, and alternative architectures.

AR1: Digital Logic

The idea of simple building blocks implemented in different technologies; different levels of integration. Physical considerations such as delays, fan-in, fan-out. The use of a Medium Scale Integrated (MSI) device (Programmable Logic Device, or PLD) to implement complex functions in a
single chip. Common flipflop types. Representation and tables. Clocked operation and ripple-through effects, MSI Devices, and their use in making many of the basic logic functions. Interconnection of large units.

1. Basic logic elements and switching theory; minimization and implementation of functions
2. Propagation delays and hazards
3. Technologies; types of flipflop
4. Devices (e.g., demultiplexers, multiplexers, decoders, encoders, adders, subtractors, comparators, shift registers, counters, PLD-type devices)
5. Memories (e.g., ROM, PROM, EPROM, EAROM, RAM)
6. Analysis and synthesis of synchronous circuits, asynchronous vs. Synchronous circuits

AR2: Digital Systems
The transfer of information from one storage device to another and the means of controlling data flow. The electronic functions of tristate devices, the bus structures and data control concepts. Various ways for describing designs.

1. Register transfer notation, conditional and unconditional
2. Algorithmic state machines, steering networks, load transfer signals
3. Tristates and bus structures
4. Iteration, top down/bottom up, divide and conquer
5. Decomposition, trade-offs, economics
6. Block diagrams, timing diagrams, transfer language

AR3: Machine Level Representation of Data
Basic machine representations of numeric and non-numeric data.

1. Numeric data representation; e.g., binary, octal, hexadecimal, fixed point, 1s and 2s complement, signed, floating point, decimal, BCD, XS3
2. Non-numeric data; e.g., alphanumeric, ASCII, ISO

AR4: Assembly Level Machine Organization
Comparisons of different types of instruction sets and corresponding addressing modes. Emphasis on the relationships among instruction sets, fetch and execute operations, and the underlying architecture. Introduction to the concept of interrupts, as well as the purpose and specifications of a control unit with respect to logic operations. Hardwired and microprogrammed control units, their respective advantages and disadvantages. Vertical and horizontal microcoding. General methods for designing for maintenance, such as breaking up the design for easy maintenance and adding extra hardware for easier access to special registers.

1. Basic organization; von Neumann, block diagram, data paths, control path, functional units (e.g., ALU, memory, registers), instruction cycle
2. Instruction sets and types
3. Assembly/machine language
4. Addressing modes (e.g., direct, indirect, register, displacement, indexing)
5. Control unit; instruction fetch and execution, operand fetch
6. I/O and interrupts
7. Hardwired realization
8. Microprogrammed realization; formats and coding

AR5: Memory System Organization and Architecture
Consideration of the physical implementation of large memory systems, together with the techniques of data storage and checking. Overall concepts of virtual memory, cache memory, and the consequences of multiprocessor/multicache architectures. Detailed discussion of the DMA process, as well as techniques for fault handling and factors affecting reliability.

1. Storage systems and technology
2. Coding, data compression, data integrity
3. Space allocation, hierarchy
4. Main memory organization, bus operations, cycle times for selection and addressing
5. Cache memory, read/write
6. Virtual memory
7. Bussing systems, control, DMA
8. Fault handling, reliability

AR6: Interfacing and Communication
Input/output control and how it is achieved. Techniques for interrupt handling.

1. Input/output control methods, interrupts
2. Interrupt acknowledgment
3. Synchronization, open loop, handshaking
4. External storage, physical organization and drives

AR7: Alternative Architectures
Comparison of stack, array, vector, multiprocessor, hypercube, RISC, and CISC machines. Introduction to the general topic of parallel architectures.

1. Comparisons
2. CISC, RISC
3. Parallel architectures (e.g., VLIW, SISD, MISD, SIMD, MIMD)
4. Tight coupling