Computer Architecture Instruction at the University of Michigan

Gary S. Tyson
Steve Reinhardt
Trevor Mudge
The University of Michigan
tyson,stever,tnm@eecs.umich.edu

1 Curriculum Overview

The Electrical Engineering and Computer Science department (EECS) confers BS, MS and PhD degrees in both electrical and computer engineering within the College of Engineering. In addition, the EECS department supports majors in the Computer Science degree program administered through the College of Literature, Science, and the Arts. Undergraduate Computer Engineers and Computer Science majors take similar courses in computer architecture; CS majors are required to complete a three course sequence (EECS 100, EECS 270 and EECS 370), with CE majors taking an additional course (EECS 373) [1].

In this paper, we will describe both undergraduate and graduate course offerings in computer architecture and discuss new trends in instruction of this material at the University of Michigan.

2 Undergraduate Education

In the past, undergraduate students in CS or CE have taken two or three required courses in computer hardware/architecture. In the past two years, we made significant changes to the introductory course to incorporate a broad overview of the computing discipline including machine organization.

In this section, we will examine the changes to the introductory course (EECS 100) as well as provide a brief description of the standard architecture sequence. The catalog description for both undergraduate and graduate courses can be found on this last page of this paper. This table also includes prerequisite requirements for each course and the average annual enrollment at the University of Michigan.

[1] Undergraduate courses are numbered from 100 to 399; graduate courses are numbered from 500-799; transitional courses offered to advanced undergraduates and first year graduate students are numbered 400-499.

2.1 EECS 100

EECS 100 is an introductory class to the Computer Science and Computer Engineering programs. Over the past two years the EECS 100 content has changed dramatically. Many introductory computer science courses are all about programming. We consider this to be unfortunate because there is much more to computer science than programming. There is the mystery of how a computer was designed, how it works, and how we get it to do our bidding. In this course, we address a broad foundation of topics in computer science to lift the veil of mystery that pervades much of the undergraduate experience. Course topics range from the components that make a computer work (instructions and gates) to high level language abstractions. A detailed description of the subject material is included in Table 1. The course is partitioned into two major areas with computer organization covered in the first half of the semester and high level language (C) organization in the later half. A large amount of material is covered in this course, but our experience has shown that high retention is possible even for the average student.

Course information for this class (and most other classes) is available on the departmental web site. These pages are generally used to improve communication among the instructor(s), teaching assistants and students in the course; the web pages are also available to the general community. the EECS 100 web page is located at http://www.engin.umich.edu/class/eecs100/ (in May 1998). A new textbook [2] has been developed to support this new approach.

2.2 EECS 270

EECS 270 is a first course on digital logic. Like similar courses at most universities, this course covers a

[2] "From Bits and Gates to C and Beyond" by Yale N. Patt and Sanjay J. Patel

Wk	Course material Covered
1	Administration, Bits and operations on bits.
2	Bits and operations on bits (continued), Basic logic structures (transistors, gates, truth table representation), ADDER, MUX, DECODER
3	Storage elements (register, memory addressing), Introduction to the von Neumann model. Emphasis on memory.
4	ISA Specification of simple processor (instruction formats, control, datapath), Problem Solving and On-Line Debugging,
5	control structure of a stored program (sequential, conditional, iteration), Assembly Language (translation, hand assembly of sample programs)
6	Physical I/O (Keyboard Data and Status Registers, Polling, Interrupts), Traps, ISRs
7	Subtroutines (JSR/RET mechanism) Stacks, Parameters. How are they passed?
8	Motivation for programming at a higher level, An introduction to the C programming language, Variables (types, declaration, scope, symbol table, allocation).
9	Operators (arithmetic, bitwise, logical, relational, assignment), Control structures (if, if-else, switch, for, while, do-while), Translation to assembly.
10	Function (prototypes, definition, calls), Revisiting the compiling, linking process; what happens with functions. Translation to assembly. Stack management.
11	Source Level Debugging, Recursion, Pointers.
12	Arrays in C, I/O in C, Structures in C, Translation to assembly.
13	Dynamic allocation, linked list data structure.
14	Analysis of algorithms (Big-O notation), Divide and conquer algorithm (binary search), Space complexity: Memory usage.
15	What comes after EECS 100? What do computer engineers and computer scientists do?

Table 1: EECS 100 Lectures.

variety of design components. Boolean algebra is first introduced in this course; logic gates, logic minimization and standard combinational circuits are presented and reinforced in a lab component to the course. Laboratory includes hardware design and CAD experiments.

2.3 EECS 370

EECS 370 has been the first course on computer organization, but with the changes to EECS 100 the content of EECS 370 is being expanded. This course focuses on pipelined architectures with students building simulators of the pipeline execution at varying levels of complexity. Both C and Verilog have been used to construct the detailed simulation.

2.4 EECS 373

While EECS 370 treats the implementation of processors from low-level digital components, EECS 373 covers the application of microprocessors as components in complete digital systems. Students learn hardware and software aspects of interfacing devices to modern microprocessors through a combination of lectures and laboratory exercises. Lectures stress fundamental, device-independent concepts, illustrated by a range of current and historical examples. Topics include bus protocols, interrupt structures, DMA, memory technologies, A/D and D/A conversion, and basic video and disk interfacing. Both embedded and general-purpose system issues are covered.

Laboratory exercises reinforce these concepts via hands-on experience with specific devices. For Fall 1998, we are introducing new, state-of-the-art lab equipment. Each lab station is built around a Motorola embedded PowerPC development system coupled with a custom interface board. Each interface board contains two Xilinx field-programmable gate arrays (FPGAs) and a number of peripherals, including switches, a seven-segment LED display, A/D and D/A converters, and a small SRAM. Students design interface circuits for the various peripherals and download them to the FPGAs. They also develop small assembly-language programs to exercise the peripherals. Students develop and debug their software using an integrated environment from Software Development Systems, Inc. (SDS) that includes a C compiler, an assembler, and a debugger. The debugger uses the Motorola processor's built-in debug facilities to achieve in-circuit emulation capabilities without additional hardware. Hewlett-Packard 16600A 136-channel logic analyzers work in tandem with the SDS debugger to provide a complete, sophisticated view of hardware ac-

tivity supporting both software and hardware debug. Both the Xilinx development tools and the SDS environment include simulation capabilities, and are available on machines in the public engineering computer labs. Students can design and test their hardware and software independently outside the lab, leaving precious lab time to focus on hardware/software integration and to observing and measuring the physical system at work.

3 Graduate Education

The University of Michigan has a very large and active graduate program, particularly in computer architecture. With over 50 PhD students associated with the Advanced Computer Architecture Lab (ACAL), there is a great demand for a variety of courses in architecture at the graduate level. All graduate students in Computer Engineering are required to take EECS 470 (or equivalent). In addition, several elective courses are regularly offered (EECS 570, 571, 573 and 583). Finally, experimental courses (EECS 598 and EECS 670) are often offered – these courses are selected to cover current topics and are therefore unlikely to be repeated.

3.1 EECS 470

The introductory graduate-level class in computer architecture (EECS470: Computer Architecture) is designed for the student who wishes to obtain a detailed understanding of how computers are designed and implemented. We assume that the entering student understands the concepts of assembly language, machine language, ALU design, and the basic ideas of pipelining, caches, and virtual memory, which are covered in our senior-level class EECS 370. EECS 470 expands on this material in both depth and breadth. In addition it introduces advanced material on instruction set design, microprogramming, instruction-level parallelism, vector processing, and storage systems. Concepts are illustrated with historical and state-of-the-art systems.

The class is offered in both the Fall and Spring semesters and meets twice a week for 90 minutes. Excluding classes taken for exams and reviews, there are about 25 classes. The class material emphasizes a quantitative approach to design. Design goals are usually formulated in terms of performance or power measures and require the students to understand performance evaluation techniques. So that the student is aware that performance comes at a cost, tradeoffs in design and implementation are discussed. Trends that will affect these tradeoffs in future systems are also discussed.

To give students a deeper understanding of the practicalities of many of the concepts and tradeoffs studied during lecture, students are required to design a substantial, realistic processor using the Verilog hardware design language. A weekly one hour discussion session is scheduled to learn how to use the design tools and to discuss implementation issues. The discussion is also used to discuss homework problems.

For the project the student is expected to use the extensive design environment that has been developed over the past 10 years at Michigan. It includes modern commercial CAD tools from Cadence, Mentor, Cascade and Avant. Not all of these tools are necessary in this particular class where the goal is to design and test a mixed behavioral/structural Verilog model that can be synthesized using the Synopsis synthesis tools. The project represents a significant investment of time for the student and the grade breakdown for the class reflects this.

The required text is the second edition of "Computer Architecture: A Quantitative Approach," by Hennessy and Patterson. Additional texts that are referred to include, "Computer Architecture: Pipelined and Parallel Processor Design," by Flynn, and the second edition of "Computer Architecture and Organization," by Hayes. Papers that supplement the text are also used and handed out in class as needed. Verilog material includes tutorial chapters from the Cadence manual and a set of class notes on synthesizeable design with Verilog.

3.2 Graduate Electives

In addition to the required graduate architecture course, electives are available to further explore different aspects of architecture. EECS 570 concentrates on the design and use of parallel processors. Students taking this course are exposed to a variety of parallel architectures (IBM SP2, HP Convex Exemplar SPP-1000, GSI Power Challenge GR, etc.). Many of the students in this course continue working on parallel machines with the Parallel Performance Project (PPP) and/or the Center for Parallel Computing (CPC) at the University of Michigan.

The EECS571 course is intended to cover principles and foundations of real-time computing (RTC). Due to its vital role in almost all application domains, such as multimedia, virtual reality, telecommunications, industrial automation, aerospace, embedded commercial and defense systems, medical instrumentation and life support systems, RTC has become an essential discipline in the field of computer science and engineering. RTC as studied in this course is based on three attributes:

high performance, ultra-high reliability, and environmental interface. These three attributes are strongly coupled together by a single precious resource, time. In this course students will be exposed to the state-of-art (both analytic and experimental) research and development related to all these three attributes and their interplay.

EECS 573 provides a fundamental body of knowledge useful to grad students who plan to do research in microarchitecture. In particular the emphasis is on combining the mastery of fundamentals with critical reading and analysis and creative thinking. The course begins with four fundamental properties of microarchitecture dealing with instruction supply, instruction processing, data supply and control flow. The bottlenecks in each of the four aspects are covered. Various architectural paradigms (e.g. VLIW, Superscalar, HPS and MultiScalar) are discussed with the focus on how these paradigms try to overcome the bottlenecks described above. Finally an introduction to patent and copyright laws are discussed and the pros and cons of patenting a design are explained.

4 New Directions in Teaching Architecture

Architecture designs continue to evolve with differing tradeoffs leading to new solutions. In order to provide the flexibility in the curriculum to explore new architectural research, two courses are available: EECS 598 and EECS 670. EECS 598 is the primary mechanism for introducing new course offerings into the curriculum; these are courses that are taught first as a 598 course and later, if successful, designated a new course number and added to the regular curriculum. In contrast, 670 courses are designed to enable instruction on a current topic one time only.

In a recent EECS 598 course, we introduced a new course examining the relationship between compiler optimization, instruction set design and microprocessor architecture. With the advent of just-in-time (JIT) compilation and object code translation, the time has come to more closely integrate compiler development with instruction set design and microarchitecture. This course has been taught once as an EECS 598 course with the intention of incorporating it as the regular graduate elective course in later years. By teaching it as a special projects course, we were to experiment with the class before finalizing a new course proposal. In this particular course, we were able to discuss current trends in instruction set design (e.g. Intel's EPIC architecture) while determining which features of the course capture more fundamental aspects which can be included in a more permanent course.

5 Summary

In this paper we have discussed the course framework for teaching computer architecture. The University of Michigan has a very active faculty in this area which enables us to offer a broad range of courses on this subject. The course structure has been developed to provide as much flexibility as possible to change course content as our field advances. This is best demonstrated in the major reordering of the introductory course (EECS 100), which is now better able to prepare incoming students for the rigors of the later courses. At the graduate level, the curriculum is designed to provide a core set of courses covering major aspects of computer architecture as well as enable the addition of experimental courses.

Course Title	Prereq Courses	Annual Enrollment	Course Description
100 Introduction to Computing Systems		800	How a computer works, from the machine level to high level programming. Circuits, instructions, memory, data. Assembly language. Binary arithmetic, data types, data structures. Translation of high level languages. The C programming language: data structures, control, iteration, recursion. Basic algorithm analysis.
270 Introduction to Logic Design	100	450	Binary and non-binary systems, Boolean algebra digital design techniques, logic gates, logic minimization, standard combinational circuits, sequential circuits, flip-flops, synthesis of synchronous sequential circuits, PLAs, ROMs, RAMs, arithmetic circuits, computer-aided design. Laboratory includes hardware design and CAD experiments.
370 Introduction to Computer Organization	270, 280	300	Computer organization will be presented as a hierarchy of virtual machines representing the different abstractions from which computers can be viewed. These include the logic level, microprogramming level, and assembly language level. Lab experiments will explore the design of a microprogrammed computer.
373 Design of Microprocessor Based Systems	370	240	Principles of hardware and software microcomputer interfacing; digital logic design and implementation. Experiments with specially designed laboratory facilities. Introduction to digital development equipment and logic analyzers. Assembly language programming. Lecture and laboratory.
470 Computer Architecture	370	90	Basic concepts of computer architecture and organization. Computer evolution. Design methodology. Performance evaluation. Elementary queueing models. CPU architecture. Introductions sets. ALU design. Hardware and microprogrammed control. Nanoprogramming. Memory hierarchies. Virtual memory. Cache design. Input-output architectures. Interrupts and DMA. I/O processors. Parallel processing. Pipelined processors. Multiprocessors.
473 Advanced Digital System Design	373	—	This course introduces advanced digital system design concepts, such as timing analysis, reliability, and testability. These concepts are then applied to a semester-long design project of the student's choice. The result of this project will be a highly testable, highly reliable digital system.
570 Parallel Computer Architecture	470	20-40	Pipelining and operation overlapping, SIMD and MIMD architectures, numeric and non-numeric applications, VLSI, WSI architectures for parallel computing, performance evaluation. Case studies and term projects.
571 Principles of Real-Time Computing	470, 482	< 20	Principles of real-time computing based on high performance, ultra reliability and environmental interface. Architectures, algorithms, operating systems and applications that deal with time as the most important resource. Real-time scheduling, communications and performance evaluation.
573 Microarchitecture	470	< 20	Graduate level introduction to the foundations of high performance microprocessor implementation. Problems involving instruction supply, data supply, and instruction processing. Compile-time vs. run-time tradeoffs. Aggressive branch prediction. Wide-issue processors, in-order vs. out-of-order execution, instruction retirement. Case studies taken from current microprocessors.
583 Advanced Compiler Construction	470, 483	< 20	Code Generation and Optimization: Advanced code generation techniques. Representation of intermediate code. Data flow analysis, code movement, loop optimization, common subexpression elimination, and peephole optimization. Optimization by program transformation. Machine specific optimizations, improving instruction schedule for pipelined microprocessors.
598 Special Topics: Trends in Instruction Set Architecture Design		varies	Trends in Instruction Set Architecture Design (taught Winter 1998): This course will analyze current trends in the evolution of instruction set architectures. In particular, we will explore modifications to existing instruction sets that enable the compiler to convey additional information useful in achieving high levels of instruction parallelism. These modifications will include: memory prefetch, speculative load, predicated execution and newer VLIW designs including the Intel/HP Explicitly Parallel Instruction Computing (EPIC) approach used in the next generation platforms for Intel (P7 or Merced) and HP. Current research projects will also be discussed.
670 Advanced Topics in Computer Architecture	570	—	Advanced concepts and specialized areas in computer systems design are discussed and analyzed in depth. Topics chosen by instructor. Examples are database machines, highly reliable systems, computers for artificial intelligence, architectural support for operating system functional, high-level language architectures, object-oriented architecture, other special purpose architecture (vision, dataflow).

Table 2: EECS Catalog Course Descriptions.