Computer Architecture Instruction for Students from Technically Diverse Backgrounds

James J. Alpigini
Penn State Great Valley School of Graduate Professional Studies

Abstract

The Master of Science in Information Science degree program at the Penn State Great Valley School of Graduate Professional Studies offers a blend of computer engineering, software engineering and management courses. These courses balance information science and management theories and allow a student to develop technical competence, leadership skills, and business expertise. A required foundation course in this program is CSE 431: Introduction to Computer Architecture, which is offered at the level of a senior undergraduate or first year graduate. Despite its fundamental nature, teaching the course represents a major challenge for the instructor due to the technically diverse background of the students, many of whom have non-technical backgrounds such as business or human services. Within this paper, the approach used to teach computer architecture to such a mixed student background is presented. Emphasis is given to the course content and tools utilized, as well as efforts to make the course challenging for the student, regardless of the student's technical level.

I. Introduction

The Master of Science in Information Science (MSIS) is a relatively new and exciting degree program at the Penn State Great Valley School of Graduate Professional Studies. The MSIS program offers a synergistic blend of computer engineering, software engineering and management courses, which emphasize a balance of information science and management theories and thus enable the development of a students technical competence, leadership skills, and business expertise.

The MSIS program is organized as follows. After completing any necessary prerequisite coursework, a student begins the program by taking sequences of engineering and management core courses. These classes emphasize the fundamental aspects of information science, in terms of both theory and practice. Once the core courses are completed, a student will take a sequence of three core electives, selected from the both the management and engineering faculties. Next, a student will participate in a "capstone" course that is offered in a seminar format and encapsulates the entire program. Following the completion of the capstone course, a student will either write a professional paper, or take three additional approved technical electives. The degree requirements are summarized in Table 1.

Table 1: Master of Science in Information Science Degree Requirements

Management core courses	MGMT 501 Behavioral Science in Business MSIS 510 Statistical Analysis for Managerial Decision Making M I S 531 Management Information Systems
Engineering core courses	CSE 428 Applied Programming Languages CSE 431 Introduction to Computer Architecture CSE 497 Introduction to Software Engineering
Core Electives (9 credits; select three courses, taking at least one from each program)	<u>Management core electives</u> B A 517 Communication Skills for Management B A 555 Business Environment MGMT 558 Seminar in Organizational Decision Making M I S 538 Decision Support Systems <u>Engineering core electives</u> CSE 465 Data Structures and Algorithms CSE 597 Advanced Software Engineering I E 514 Data Management Systems Design M I S 537 Management Information Systems Design
Capstone course (3 credits; taken as 10th course)	M I S 539 Management of M I S
Take either option A or B	<u>Option A – Paper option</u> ENGR 594 Master's Professional Paper (3 credits) <u>Option B – Elective option</u> (9 credits; select three courses from approved I.S. curriculum)

A tenet of the MSIS program is that to manage technology effectively, it is necessary first to understand the technology. For this reason, a course in computer architecture is included as one of the engineering core courses. The course is introductory in nature, offered at the level of a senior undergraduate or first year graduate. Despite its fundamental nature, teaching the course represents a major challenge for the instructor due to the diverse technical backgrounds of the students, many of whom are "career shifters." That is, they are shifting into the field of information science from non-technical backgrounds such as education, business, or human services.

Within this paper, the approach used to teach CSE 431: Introduction to Computer Architecture to such a diverse student background is presented. Emphasis is given to the course content and tools utilized, as well as efforts to make the course challenging for the student, regardless of the student's technical level. The paper is organized as follows. First, the course objectives are presented with a description of the challenges faced in its instruction. Next, the organization of the course is presented; giving emphasis to a project designed to challenge the student, regardless of his or her technical background. Software developed to assist in assembly instruction is next considered and then the paper concludes with some remarks on the use of technology in future computer architecture course offerings.

II. Course Objectives

CSE 431: Introduction to Computer Architecture is normally taken by a first year MSIS student and requires no prerequisite beyond graduate standing. The primary objective of the course is to provide the student with the necessary background with which to complete the technical portion of the MSIS degree, as well as provide the fundamental knowledge to succeed in the information technology (IT) field. In response to the needs of the MSIS program, the course emphasizes architectural software, that is, assembly code, and the logical, organizational design of a computer. Thus, there is no need for an electronics background to succeed in the course.

In keeping with the standard course offering at Penn State Great Valley (PSGV), CSE 431 is organized in a seven-week format with three hour class meetings held two nights per week. The PSGV student body favors this compressed format because it enables the completion of a Master's degree in an average of three years, while still taking just one course at a time. The average class size is 30 to 35 students, of which over 90% are working professionals who are pursuing a Master's degree in the evening. The average incoming undergraduate grade point average is quite high, being greater than 3.0 out of 4.0, but the technical background of students is widely varied. Moreover, many of the students have been away from school for long periods of time, often decades. The combination of this varied background and the seven-week course compression serves to make the instruction of CSE 431 a challenging task.

III. Course Organization

A listing of the typical syllabus for the class is given in Table 2. Presenting a hierarchical view of architecture, the course emphasizes the complex instruction set computer (CISC) architecture in general, and the Von Neuman paradigm in particular. Wherever appropriate, the instruction focuses on the implementation of the Intel Pentium™ microprocessor and the PC style computer. This has been found to be of the most interest to the students, as they commonly own such a device. Because of the course time compression, many topics which could easily have been covered in one evening are spread over two or more evenings. While this can disrupt a sequential flow of topics somewhat, it has been found to be a better approach. Splitting subject matter, which can be somewhat dry, into smaller portions has a positive effect on student attention and also serves to reinforce concepts as they are revisited in subsequent lectures.

Although the Pentium™ is frequently used to exemplify architecture, the course emphasizes that any computer architecture represents a series of design tradeoffs, and that there is no perfect architecture. When reduced instruction set computer (RISC) architectures are discussed, an emphasis is given to the incorporation of RISC elements in modern CISC architectures. There are many exercises that are presented *in-class*, during which the instructor assists students as needed. This serves to immediately enforce the topics covered and greatly assists students with difficult material.

Table 2: Topical organization of the course

Lecture #	Topics Covered
1	Overview; Von Neuman machines; Instructions and Fetch-Defer-Execute cycle; Merit and benchmarks
2	Register sets; Computer math; Data handling
3	Floating point conversion; Instructions; Assembly code, part 1
4	Assembly code, part 2; Memory addressing schema, part 1
5	Memory addressing, part 2; Assembly code, part 3; Memory design issues
6	Mid Term Examination
7	Bus architecture; CPU Organization and Micro Programs; Machine Startup
8	Exception processing; Input/Output; Memory Chips
9	Memory systems and interleaving; Cache memory
10	Virtual memory systems; RISC machines
11	Pipelines; Superscalar processing
12	Project Night - No formal class
13	Client Server; Inter-Computer Communication
14	Final examination

The grading scheme for the course is 30% mid-term examination, 40% final examination and 30% project. Although it comprises less than a third of the numerical score for the course, a major emphasis is given to the project, including portions of in-class time. A formidable task in the course instruction is to challenge the more advanced student, while enabling the student with little to no technical background to master the material. For this reason, the students may select a project of their choice from the list shown in Table 3. It has been found that the students will generally attempt the most advanced project of which they feel capable. In fact, it is rare to observe students select a project simply because they find it easy. The majority of students perform one of the Pentium™ comparisons, which again reflects their personal interest in the architecture. Students without any technical background tend to gravitate toward project 1, which is to compare the architectures of the Intel 8086 and Motorola 68000 microprocessors. While dated, these processors contain many elements of the newest chips and aptly reinforce the concept of design tradeoff. The more advanced students, typically with some type of computer science or software background, perform one of the Kermit simulator projects. Kermit is a simplified computer architecture employed to teach the instruction set architecture (ISA) and the concepts of assembly programming. Kermit is discussed in some detail in section 4.

Table 3: List of course projects

Project #	Project Description
1	Compare and contrast the Intel 8086 microprocessor with the Motorola 68000 microprocessor. (Data sheets are provided by the instructor)
2	Compare and contrast the Intel 80X86 family with the Motorola 680X0 family.
3	Compare and contrast the Intel Pentium, Pentium Pro and Pentium II processors.
4	Compare and contrast the Intel Pentium, Pentium II and Pentium III processors.
5	Compare and contrast two instructor-approved microprocessor architectures of student's choice, e.g. PowerPC 601 and Alpha 21064.
6	Program a simulator of the fictitious Kermit computer. The simulation is to use *integer* math and *integer* addressing.
7	Program a simulator of the fictitious Kermit computer. The simulation is to use *hexadecimal* math and *hexadecimal* addressing.

IV. Kermit Simulator

Kermit, originally proposed by Russell & Haden [1], is a fictitious computer architecture used to teach computer architecture in general, and assembly language concepts in particular. Kermit, shown in figure 1, emulates a Von Neuman CISC machine, and contains many elements of modern computer architectures, including: CPU, memory, register set, program counter, processor status word, bus, keyboard and CRT. A small portion of memory is designated as the system stack, with one of the registers designated as a stack pointer. The instruction set is listed in table 4, and is ultimately derived from the Motorola 68000 ISA, albeit in a very simplified form.

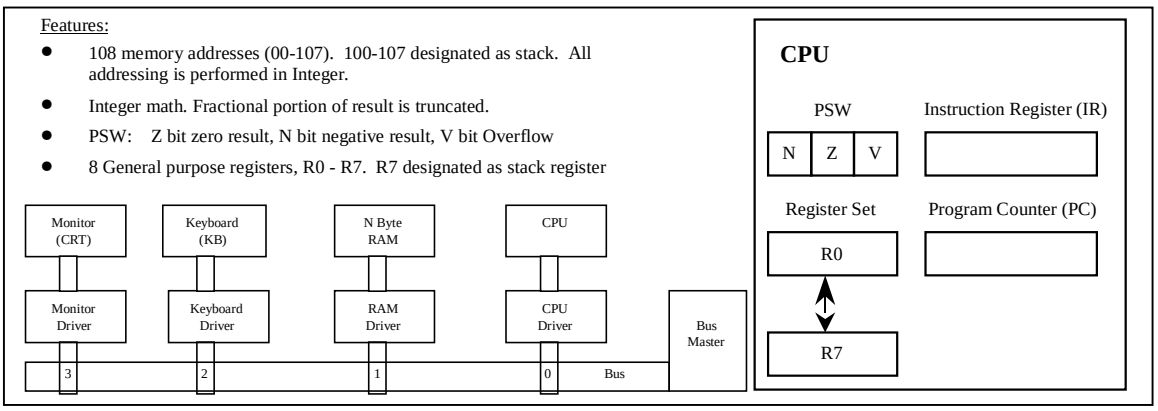

Figure 1: Kermit emulator block diagram.

Table 4 lists the instructions included with Kermit. Although it is a simple architecture, it allows conditional branching, jump and store return, basic math functions and load and store, using both direct and register indirect addressing methods. As an instructional tool, Kermit has been shown to be invaluable. For example, two topics covered are vector processing and

accumulator machines. In one demonstration, vector operations are performed in Kermit, an inherently scalar architecture. These are then contrasted with identical operations using a vector processing architecture. In another demonstration, a mathematical sequence is performed using Kermit, and then again using an accumulator. Kermit enables relatively painless understandings of what can be confusing subjects to the novice.

Table 4: Kermit instruction set

OpCode	Mnemonic	Op. 1	Op. 2	Notes
00	HLT			Immediate Stop
01	NOP			No Operation
30	CLR	Ri		Clear Ri to Zero, Set Z bit to 1, set N, V bits to 0.
31	INC	Ri		Increment Ri, Set N, Z, V bits based on result.
32	DEC	Ri		Decrement Ri, Set N, Z, V bits based on result.
50	BR	A		Unconditional Branch to Address A
51	BRX	Ri		Branch to A = (Ri)
52	BEQ	A		Branch if Z bit = 1 to A
53	BNE	A		Branch if Z bit = 0 to A
54	BPL	A		Branch if Z bit = 0 and N bit = 0 to A
55	BNG	A		Branch if N bit = 1 to A
56	BRV	A		Branch if V bit = 1 to A
57	JSR	A		(PC) → Stack; Branch to A
58	RET			Stack → PC
60	ADD	Ri	Rj	Rj ← (Rj) + (Ri). Set N, Z and V bits based on result.
61	SUB	Ri	Rj	Rj ← (Rj) – (Ri). Set N, Z and V bits based on result.
62	MULT	Ri	Rj	Rj ← (Rj) * (Ri). Set N, Z and V bits based on result.
63	DIV	Ri	Rj	Rj ← (Rj) / (Ri). Set N, Z and V bits based on result.
64	CMP	Ri	Rj	Temp ← (Ri) – (Rj). Set N, Z and V based on Temp
70	LDI	Ri	N	Load N into Ri. Set N, Z and V bits based on result.
71	STO	Ri	A	Store (Ri) into A. Set N, Z and V bits based on result.
72	STX	Ri	Rj	Store (Ri) into A = (Rj). Set N, Z,V bits based on result.
73	LD	Ri	A	Load (A) into Ri. Set N, Z, V bits based on result.
74	LDX	Ri	Rj	Load (A = (Rj)) into Ri. Set N, Z, V bits based on result.
85	CRT	Ri		Display (Ri) on CRT. Set N, Z, V bits based on result.
86	KB	Ri		Put KB value into Ri. Set N, Z, V bits based on result.
99	START	A		Start Execution at Address = A. Clear PSW

Kermit Simulator

To assist with the use of the Kermit architecture, a simulator has been coded which is used extensively in class and made available to the students for study on their home machines. The simulator was coded in Borland Delphi™ using an intuitive graphical user interface. Instructions are executed manually, with depression of the “Step” button cycling through the individual fetch, defer and execute (FDE) phases of each instruction. The simulator allows all memory locations and registers to be viewed by the user. Keyboard and CRT instructions are handled in “pop-up” windows. A screen snapshot of the simulator is shown in Figure 2.

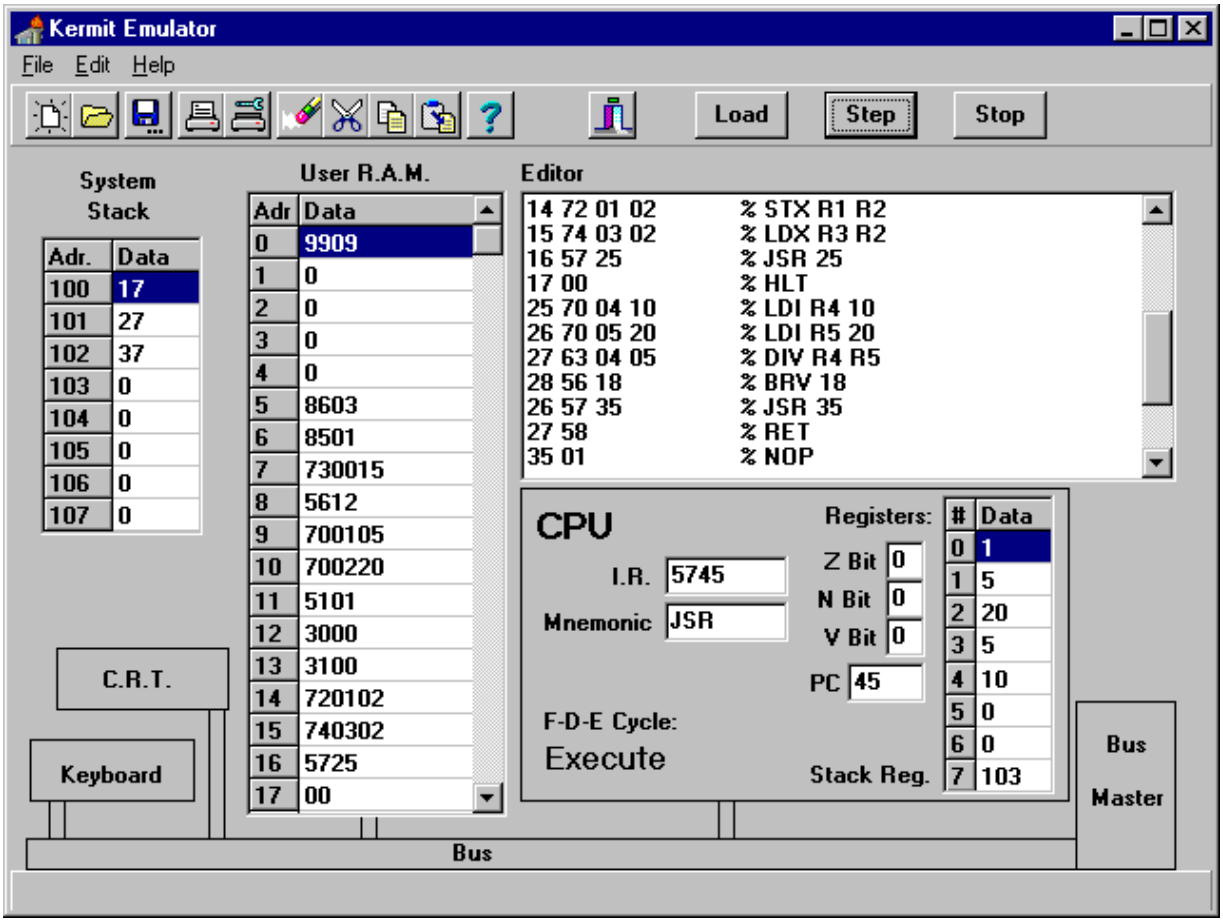

Figure 2: Kermit emulator screen shot

It is indeed possible to teach ISA concepts without using a simulator. However, by allowing the students to execute their programs and immediately view instruction results, the use of a simulator better facilitates an understanding of the subject [2]. Moreover, because the addressing and mathematics are performed solely in integer mode, the simulator is somewhat more intuitive than were it to employ hexadecimal addressing or math. Another simplification employed is that of the address size. Each memory address holds one instruction or data element, regardless of the size of the stored item. While this is a departure from a true computer architecture, it facilitates an easier understanding of the defer portion of the FDE cycle.

V. Discussion

In this paper, an approach to computer architecture instruction for a student body of diverse technical background was described. This approach has been proven quite successful, with over twenty sections of the course taught to date. The course is continually evolving, with many innovations being the result of student feedback. One such innovation has been the conversion of all lecture notes to Microsoft PowerPoint™ presentations. This enables the slides to be delivered using a Proxima computer screen projector and also for hard copies of the slides to be

distributed. The result is that classes require less time to cover material, enabling more in-class time for project work.

The current direction of the course is to increase the use of instructional technology in its offering. All course handouts are now available to students via the Internet. Currently, the use of an Internet chat room to augment the course instruction is being experimented with. The instructor is "on-line" for all scheduled web chats. Because the students are working professionals, it is inevitable that lectures are occasionally missed. The chat room allows students on travel to participate in a class as well as affording instructor assistance as needed.

In summary, CSE 431: Introduction to Computer Architecture is offered at the Penn State Great Valley School of Graduate Professional Studies as part of the Master Of Science in Information Science. The target student body comes from very diverse technical backgrounds, increasing the challenges in the course instruction. The course is continually evolving. It incorporates a self-scaling project, which enables students to select the project difficulty that they feel most able to complete. The course employs Kermit, a simulated computer, to reinforce computer architecture in general, and the instruction set architecture in particular. The course has been very successful, both in terms of student feedback and in their successes in the remainder of the MSIS program.

Bibliography

1. Russell, D.W., Haden, K.B, (1990), A Configurable, Virtual Microprocessor System for Instructional Use in Real Time Real World Systems, *Proceedings of the twenty-first Annual Pittsburgh Conference on Modeling and Simulation*, 5/3/90-5/4/90, V3 Part 2, pp. 1181-1185.
2. Rotithor, H.G., (1995), On the Structure of a Multipurpose Introductory Course in Computer Architecture at the Worcester Polytechnic Institute, *IEEE Transactions on Education*, V 39, No. 3, August 1995, pp. 230-235.

JAMES J. ALPIGINI
James J. Alpigini is an Assistant Professor of Systems Engineering at the Penn State Great Valley School of Graduate Professional Studies. In addition to research, he teaches in the areas of computer architecture, computer security, numerical analysis and mechatronics. He received a B.E.E. degree from Villanova University in 1982, a M.Eng.E.S. degree from the Pennsylvania State University in 1993 and a Ph.D. from the Engineering Faculty at the University of Wales, Swansea in 1999.