The Relatively Simple CPU Simulator

John D. Carpinelli
New Jersey Institute of Technology

Abstract

The Relatively Simple CPU Simulator is an instructional aid for students studying computer architecture and CPU design, typically at the junior or senior level. It simulates the Relatively Simple CPU, a 16-instruction processor introduced in the textbook *Computer Systems Organization and Architecture*[1]. Students first enter an assembly language program, which is assembled by the simulator. After correcting any syntax errors, the user simulates the fetch, decode, and execute cycles for each instruction in the program. The user may simulate the execution of the program by clock cycle, by instruction, using breakpoints, or as a single, continuous execution.

The simulator uses animation to give students a more intuitive understanding of how the CPU fetches, decodes, and executes instructions. It shows the flow of data within the CPU's register section. A pop-up window animates data flow within the ALU whenever it is active. The control unit highlights signals asserted by the control unit and used in the rest of the CPU. Users may select either a hard-wired or microcoded control unit.

The Relatively Simple CPU simulator is coded as a platform-independent Java applet that can be executed within any Java-enabled web browser. The simulator and its source code are freely available under the GNU Public License.

1. Introduction

The goal of this simulation package is to actively engage students in the process of learning how a CPU works. Students who take a passive approach to learning are less likely to learn the material and are less likely to perform well in their courses. By illustrating the flow of data within a CPU as it fetches, decodes, and executes instructions, this simulator will help students to learn the material better.

Most textbooks for computer organization and architecture have some type of simulator available[2,3,4]. (One notable exception[5] does not offer a simulator.) However, these simulators only accept program input and output results, such as the contents of registers after each instruction. They show students what happens within a computer, but not the actions that cause each operation to occur. They do not show how data moves from one place to another, only that it does so.

The Relatively Simple CPU Simulator uses visualization to illustrate the flow of data between components in a CPU. Animating the flow of data within the system provides students with a more intuitive understanding of how the CPU fetches, decodes, and executes instructions. As its name implies, the processor simulated by this package is relatively simple, on the order of complexity of Intel's 8085 microprocessor. However, this is sufficient to teach students the basics of CPU organization and design without burdening them with too many details.

This simulator is written in Java and is executed as an applet within any Java-enabled web browser. The primary reason for doing this was to provide platform independence. The Relatively Simple CPU Simulator can be executed on any Java-enabled web browser, regardless of the type of computer used. By excluding proprietary extensions in the source code, the simulator realizes the "write once run anywhere" mantra of Java developers. Using Java for all of the simulators developed by the authors allows maximum reuse of code. The assembler for this simulator is also used in another simulator which simulates a computer system incorporating the Relatively Simple CPU. As an additional benefit, the authors have found little difficulty in attracting students to work on this and other simulators. Java is a desirable skill for graduates entering the workforce, and students seek to gain experience in Java programming. Projects of this type are exactly what these students are looking for.

The rest of this paper is organized as follows. The specifications of the CPU simulated by this package are described in the following section. The functions of the simulator are given in the next section; finally, concluding remarks are presented.

2. CPU Specifications and Design

The Relatively Simple CPU is an 8-bit processor with a 64K address space. It interfaces to memory and I/O devices via a 16-bit address bus and an 8-bit system data bus. The Relatively Simple CPU uses memory-mapped I/O, so only Read and Write signals are included in the system's control bus. (Other control signals found in some CPUs, such as a READY signal, were excluded to simplify the presentation of the processor.)

The instruction set architecture of the Relatively Simple CPU includes three registers that can be controlled directly by the programmer. The accumulator, *AC*, is an 8-bit register. It receives the result of any arithmetic or logical operation and provides one of the operands for arithmetic and logical instructions that use two operands. Whenever data is loaded from memory, it is loaded in to the accumulator; data stored to memory also comes from *AC*. Register *R* is an 8-bit general purpose register. It supplies the second operand of all 2-operand arithmetic and logical instructions and can also be used to store data. Finally, there is a 1-bit zero flag, *Z*, which is set whenever an arithmetic or logical instruction is executed.

There are several other registers in this CPU which are not a part of the instruction set architecture, but which the CPU uses to perform the internal operations necessary to fetch, decode, and execute instructions. These registers are fairly standard, and are found in many CPUs. The Relatively Simple CPU contains the following registers.

- A 16-bit Address Register, *AR*, which supplies an address to memory via address pins A[15..0]
- A 16-bit Program Counter, *PC*, which contains the address of the next instruction to be executed or the address of the next required operand of the instruction
- An 8-bit Data Register, *DR*, which receives instructions and data from memory and transfers data to memory via data pins D[7..0]
- An 8-bit Instruction Register, *IR*, which stores the opcode fetched from memory
- An 8-bit Temporary Register, *TR*, which temporarily stores data during instruction execution

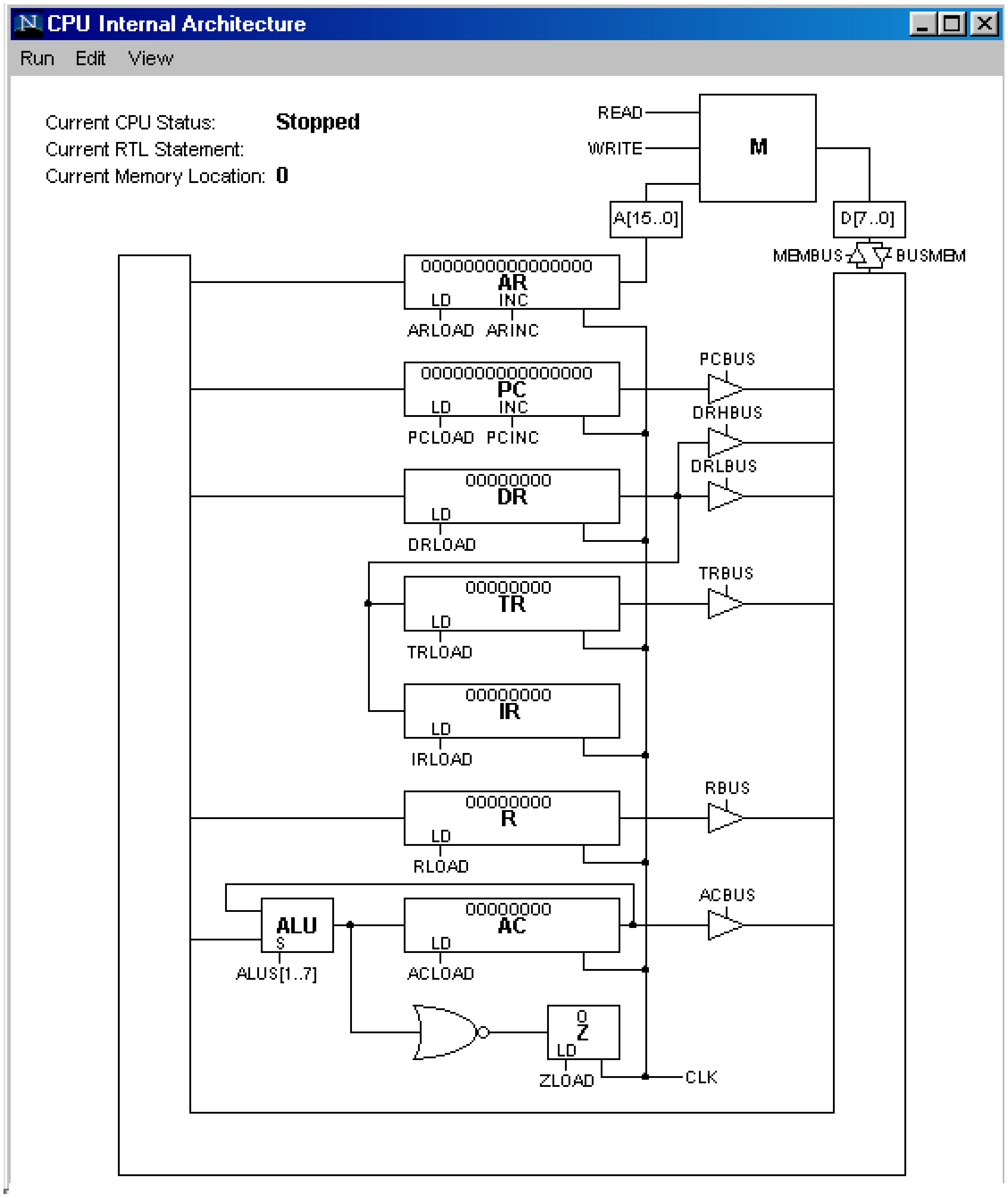
The registers within the Relatively Simple CPU are connected via a 16-bit internal bus. In addition, there are a few direct connections between some components within the CPU. (This was done to allow two data values to be transferred simultaneously.) The internal organization of the register section of the Relatively Simple CPU is shown in the screen shot of the CPU Internal Architecture window of the simulator, shown in Figure 1.

The arithmetic/logic unit for the Relatively Simple CPU is designed as two separate sections, one of which processes arithmetic operations and the other for performing logical functions. The ALU is shown in Figure 2.

The instruction set for this CPU contains 16 instructions. Although it is possible to encode these instructions using only four bits, this CPU uses an 8-bit opcode. This was done because the instruction set is expanded later in the textbook[1] as other topics, such as interrupts, are introduced. The instructions were chosen to represent instructions and instruction types commonly found in processors of this level. The instruction set for the Relatively Simple CPU is shown in Table 1.

The LDAC, STAC, JUMP, JMPZ and JPNZ instructions all require a 16-bit memory address, represented in the instruction code by Γ. Since each byte of memory is 8 bits wide, these instructions each require three bytes in memory. The first byte contains the opcode for the instruction and the last two bytes contain the address. Following the convention used by Intel's 8085 microprocessor, the second byte contains the low-order 8 bits of the address and the third byte contains the high-order 8 bits of the address.

The Relatively Simple CPU can use either a hard-wired or microcoded control unit, either of which can be simulated by this package. Figures 3 and 4 show the hardware implementations of both control units.



Current CPU Status: **Stopped**
Current RTL Statement:
Current Memory Location: **0**

Figure 1: Register section of the Relatively Simple CPU

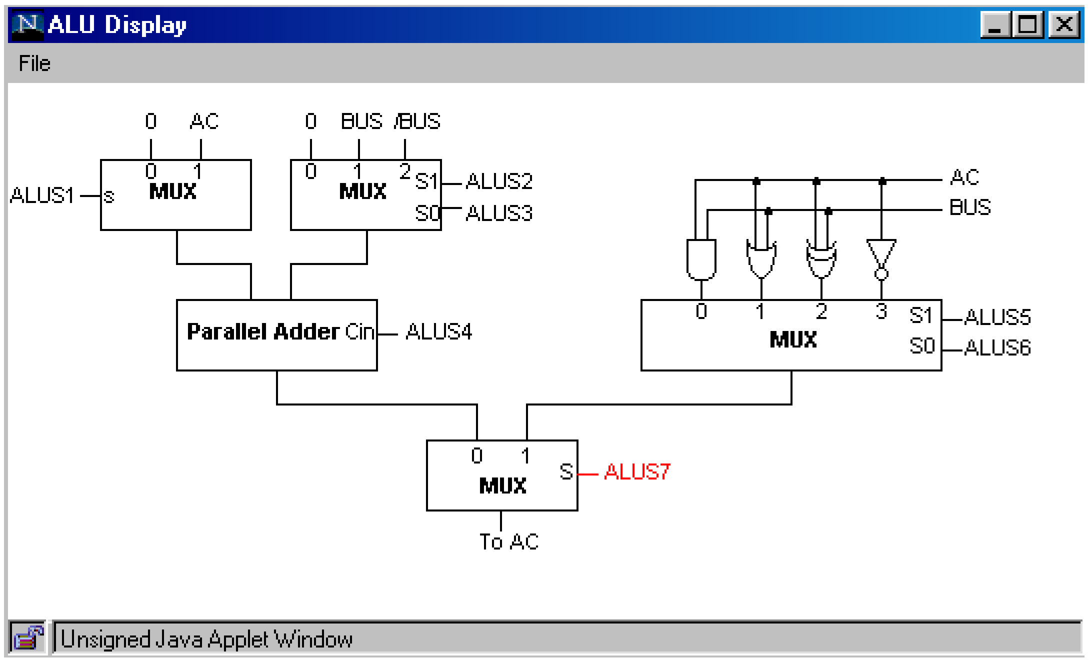


Figure 2: Arithmetic/logic unit of the Relatively Simple CPU

Instruction	**Instruction Code**	**Operation**
NOP	0000 0000	No operation
LDAC	0000 0001 Γ	*AC* = M[Γ]
STAC	0000 0010 Γ	M[Γ] = *AC*
MVAC	0000 0011	*R* = *AC*
MOVR	0000 0100	*AC* = *R*
JUMP	0000 0101 Γ	Goto Γ
JMPZ	0000 0110 Γ	IF (Z=1) THEN Goto Γ
JPNZ	0000 0111 Γ	IF (Z=0) THEN Goto Γ
ADD	0000 1000	*AC* = *AC* + *R*, If (*AC* + *R* = 0) Then *Z* = 1 Else *Z* = 0
SUB	0000 1001	*AC* = *AC* - *R*, If (*AC* - *R* = 0) Then *Z* = 1 Else *Z* = 0
INAC	0000 1010	*AC* = *AC* + 1, If (*AC* + 1 = 0) Then *Z* = 1 Else *Z* = 0
CLAC	0000 1011	*AC* = 0, *Z* = 1
AND	0000 1100	*AC* = *AC* ^ *R*, If (*AC* ^ *R* = 0) Then *Z* = 1 Else *Z* = 0
OR	0000 1101	*AC* = *AC* ∨ *R*, If (*AC* ∨ *R* = 0) Then *Z* = 1 Else *Z* = 0
XOR	0000 1110	*AC* = *AC* ⊕ *R*, If (*AC* ⊕ *R* = 0) Then *Z* = 1 Else *Z* = 0
NOT	0000 1111	*AC* = *AC*′, If (*AC*′= 0) Then *Z* = 1 Else *Z* = 0

Table 1: Instruction set for a Relatively Simple CPU

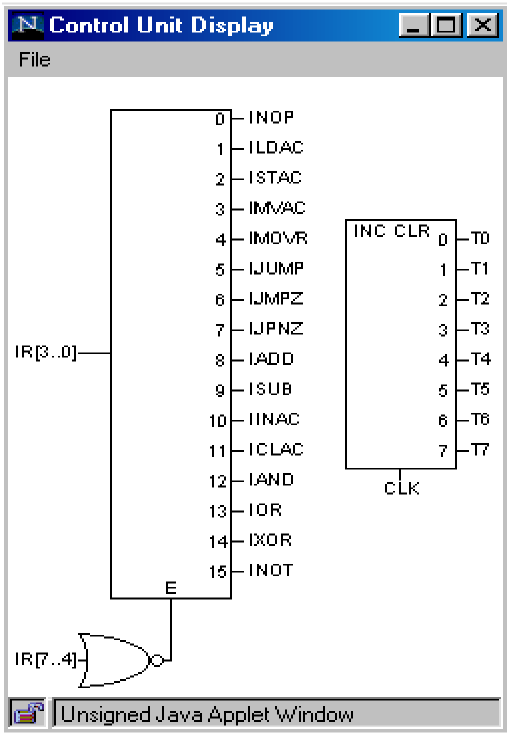


Figure 3: Hard-wired control unit of the Relatively Simple CPU

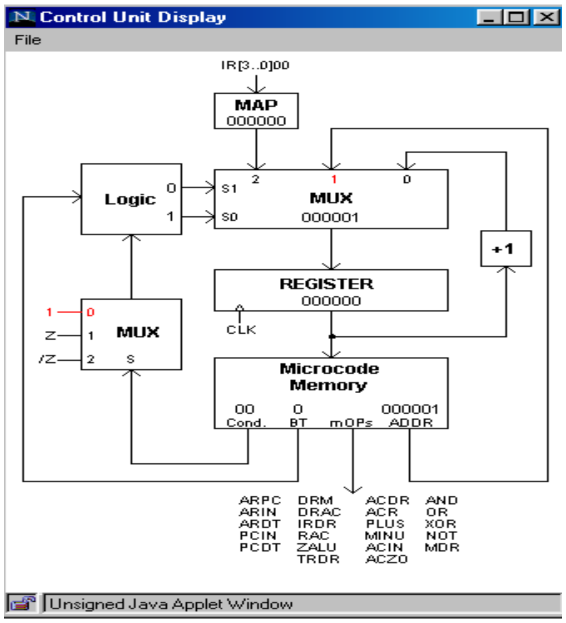


Figure 4: Microcoded control unit of the Relatively Simple CPU

3. Simulator Functions

To use the simulator, the user first starts the simulator from within any Java-enabled web browser. The simulator then presents the opening screen shown in Figure 5.

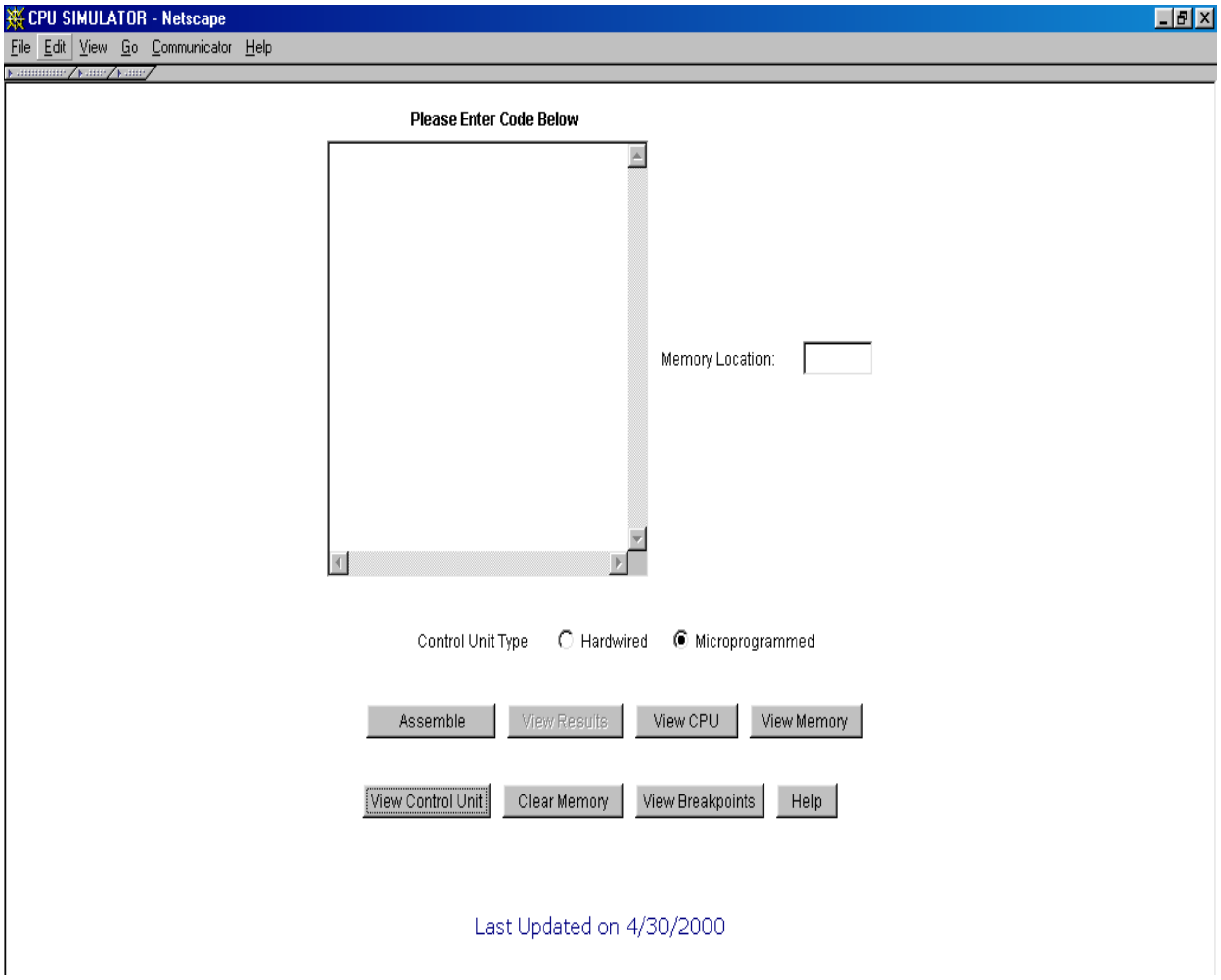


Figure 5: Opening screen of the Relatively Simple CPU Simulator

The user enters an assembly language program in the program text area and assembles the program. The simulator lists any errors encountered, which can be corrected by the user. Once the program has been assembled correctly, the user may view the contents of memory and the value at the I/O device. The user may also modify the contents of both memory and the I/O port, for example, to enter data to be used by the program. Because the Relatively Simple CPU executes its conditional jump instructions slightly differently, depending on the type of control unit used, the user may select either a hard-wired or microcoded control unit for the simulation.

After successfully assembling a program, the user executes the program from within the CPU Internal Architecture window, shown in Figure 1. As with the Relatively Simple Computer System Simulator, the user may execute the program in continuous mode, with or without breakpoints, or by single stepping through each instruction or each clock cycle. To facilitate the testing of programs for specific cases, the user may set the value of any register or memory location whenever the simulation is stopped. Thus the user may set initial conditions, or change the value of any register or memory location when the simulator is stopped between single steps or at a breakpoint. The simulator animates the flow of data between components within the CPU using dots that move along the buses and direct connections to show the direction of data flow. Active control signals for the registers are highlighted in red. The flow of data is also animated within the ALU, which is shown in a pop-up window whenever it is active. (The ALU window was implemented as a pop-up window to simplify the screen presentation of the simulator.)

During program simulation, the control unit is also simulated. For the hard-wired control unit, there is no data flow per se; the function of this control unit is shown by highlighting the active signals within the control unit. For the microcoded control unit, the flow of data within the control unit is animated, and active signals are highlighted in red.

4. Summary

The Relatively Simple CPU Simulator simulates the internal functions of its CPU as it processes the instructions in its instruction set. By animating the flow of data within the CPU, the simulator provides students with a more intuitive understanding of how the CPU fetches, decodes, and executes instructions. It serves as a useful adjunct for students using the textbook *Computer Systems Organization and Architecture*. Both the executable and source code for this simulator are available at the book's companion web sites[6,7]. The source code is available without cost under the terms of the GNU Public License.

Acknowledgments

The author thanks Aamish Kapadia, Ray Bobrowski, Leo Hendriks, and Benedicto Catalan for their efforts in coding this simulator.

Partial funding for student support for this program was provided through the Provost's Challenge Grant Program at NJIT. The authors also acknowledge the support of the Gateway Engineering Educational Coalition.

Bibliography

1. Carpinelli, John D. *Computer Systems Organization and Architecture*. Reading, MA: Addison-Wesley (2001).
2. Patterson, David A. and John L. Hennessy. *Computer Organization & Design: The Hardware/Software Interface, 2nd edition,* San Francisco: Morgan Kaufmann Publishers (1998).
3. Stallings, William. *Computer Organization and Architecture, 5th edition,* Upper Saddle River, NJ: Prentice Hall (2000).

4. Tanenbaum, Andrew S. *Structured Computer Organization, 4th edition,* Upper Saddle River, NJ: Prentice Hall (1999).
5. Mano, M. Morris. *Computer Systems Architecture, 3rd edition*, Upper Saddle River, NJ: Prentice Hall (1993).
6. URL: www.awl.com/carpinelli; Companion web site for *Computer Systems Organization and Architecture*
7. URL: www.awl.com/info/carpinelli; Companion web site for *Computer Systems Organization and Architecture*

JOHN D. CARPINELLI
John D. Carpinelli is an associate professor of Electrical and Computer Engineering, and Computer and Information Sciences, at New Jersey Institute of Technology. He received the B.E. in Electrical Engineering from Stevens Institute of Technology in 1983, and the M.E. in Electrical Engineering and Ph.D. in Computer and Systems Engineering from Rensselaer Polytechnic Institute in 1984 and 1987, respectively. Since 1986 he has been with the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology. He has served as the Associate Director and Director of Computer Engineering, and as Acting Associate Chairperson of the ECE Department. Prof. Carpinelli's research interests include interconnection networks, computer architecture, parallel processing, distance learning, and computer simulation. He has developed several simulation packages for use in undergraduate and graduate courses, both for distance learning delivery and face-to-face instruction.