THE RELATIVELY SIMPLE COMPUTER SYSTEM SIMULATOR – A VISUALIZATION TOOL FOR COMPUTER SYSTEM ORGANIZATION AND ARCHITECTURE

John D. Carpinelli
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102

ABSTRACT

Software simulation packages are available for many topics covered in undergraduate engineering programs. These tools are strong in terms of computation, allowing students to design systems and to simulate their behavior. However, most tools are not designed to help students visualize and learn basic concepts. This paper presents the Relatively Simple Computer System Simulator. This visualization tool uses animation to show how a small computer system, consisting of a microprocessor, memory, and an input/output device, functions as it processes instructions. The source and executable files for this simulator are available without cost under the terms of the GNU General Public License.

INTRODUCTION

Having students take an active role in their learning is a desirable goal for educators. Students that take an active role in their learning understand and retain their material better, whereas students who sit passively in lecture courses often have difficulty paying attention to what is being taught, and will have greater difficulty learning the material. Simulation software is one way to actively engage students in their own education. Simulation in and of itself is not a goal. The goal is to improve student learning; simulation is a strategy used to achieve that goal.

Most simulation software is designed to allow students to design and simulate systems. Consider, for example, the topics taught in a course on computer architecture. Most textbooks for computer organization and architecture have some type of simulator available.[1,2,3] (One notable exception[4] does not offer a simulator.) However, these simulators only accept program input and output results, such as the contents of memory and registers after each instruction. These simulators show students the final results of operations that occur 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. Visualization, often achieved using simple animation, is a powerful means to illustrate how a system performs its functions. For example, animation can be used to show data flowing from one system component to another, providing students with a more intuitive understanding of how a system performs its functions.

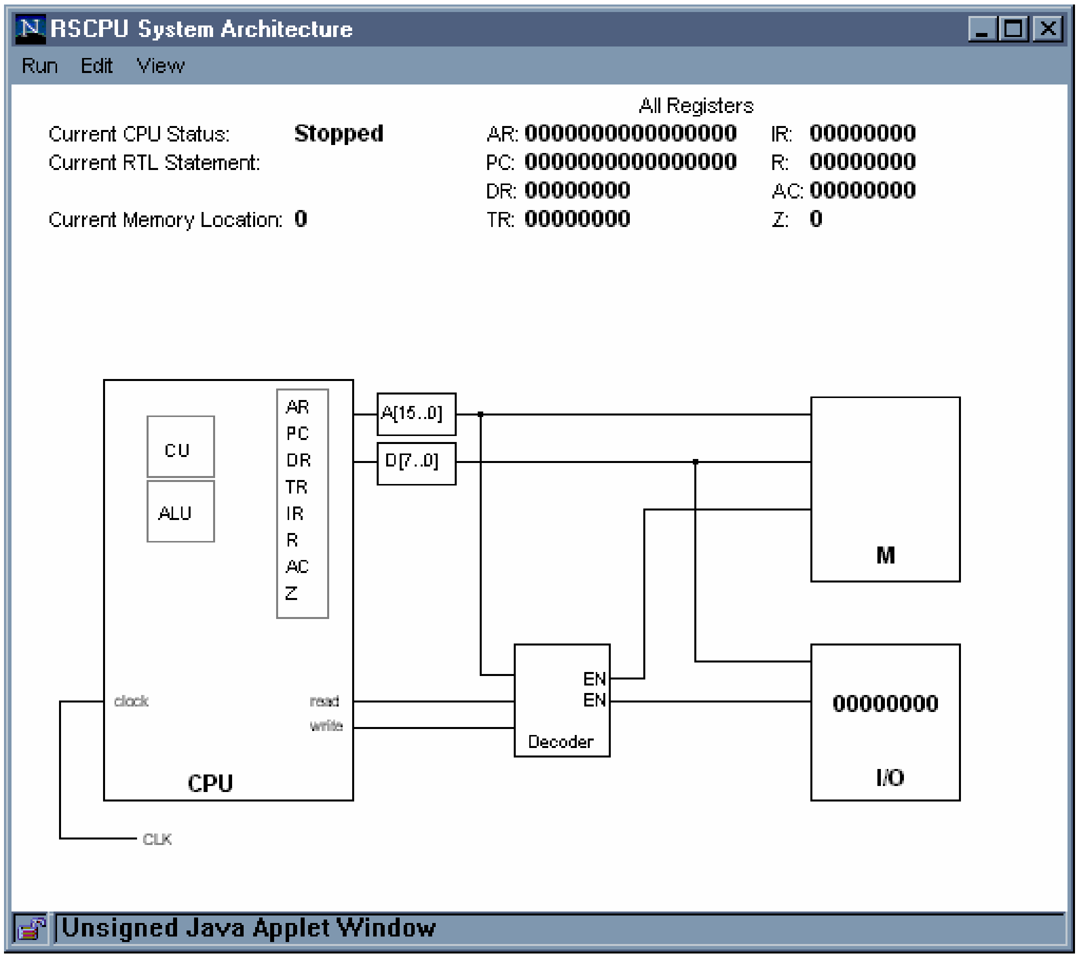
This paper describes the Relatively Simple Computer System Simulator. This visualization tool simulates a small computer system at the level of complexity that might be found in a consumer appliance. The next section describes the system that is simulated, and the following section describes the functions of the visualization tool. Other visualization tools developed through this program are also introduced.

THE SYSTEM UNDER SIMULATION

The Relatively Simple Computer System Simulator is designed to work with a computer system having a fixed architecture. Students would typically study the system on paper, and then use the simulator to enhance their understanding of how it functions. The system consists of a microprocessor, the Relatively Simple CPU (introduced as an instruction aid in the textbook *Computer Systems Organization and Architecture*[5]), 64K - 1 bytes of memory, and a bi-directional, memory-mapped I/O port at address FFFFH. This system uses a 16-bit address bus and an 8-bit data bus. READ and WRITE signals output from the CPU comprise the system's control bus. The system is shown in a screen shot taken from the simulator in Figure 1.

The instruction set architecture for this CPU includes three registers that can be controlled directly by the programmer. The accumulator, *AC*, is an 8-bit register. As in many microprocessors, the accumulator of this CPU receives the result of any arithmetic or logical operation. It also provides one of the operands for arithmetic and logical instructions that use two operands, and the sole operand for one-operand instructions. All data loaded from memory is loaded in to the accumulator and all data written to memory 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 be used to store data that the accumulator will soon need to access. Finally, there is a 1-bit zero flag, *Z*. The *Z* flag is set whenever an arithmetic or logical instruction is executed. If the result of the instruction is 0, then *Z* is set to 1 to indicate that a zero result was generated. Otherwise it is set to 0. There are no instructions which explicitly set *Z* to 0 or 1.

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 needed to fetch, decode,


RSCPU System Architecture

Run Edit View

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

All Registers

AR:	0000000000000000	IR:	00000000
PC:	0000000000000000	R:	00000000
DR:	00000000	AC:	00000000
TR:	00000000	Z:	0

Unsigned Java Applet Window

Figure 1: Relatively Simple Computer System architecture

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 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 because the instruction set is expanded later in the textbook as other topics, such as interrupts, are introduced. The instructions in this instruction set architecture 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 as 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. The low-order half of the address is stored in the second byte of the instruction code, and the high-order half is stored in the third byte.

Instruction	Operation
NOP	No operation
LDAC Γ	AC = M[Γ]
STAC Γ	M[Γ] = AC
MVAC	R = AC
MOVR	AC = R
JUMP Γ	Goto Γ
JMPZ Γ	IF (Z=1) THEN Goto Γ
JPNZ Γ	IF (Z=0) THEN Goto Γ
ADD	AC = AC + R, Set Z to 1 if result = 0 else set Z to 0
SUB	AC = AC - R, Set Z to 1 if result = 0 else set Z to 0
INAC	AC = AC + 1, Set Z to 1 if result = 0 else set Z to 0
CLAC	AC = 0, Set Z to 1
AND	AC = AC ^ R, Set Z to 1 if result = 0 else set Z to 0
OR	AC = AC $\vee$ R, Set Z to 1 if result = 0 else set Z to 0
XOR	AC = AC $\oplus$ R, Set Z to 1 if result = 0 else set Z to 0
NOT	AC = AC′, Set Z to 1 if result = 0 else set Z to 0

Table 1: Instruction set for a Relatively Simple CPU

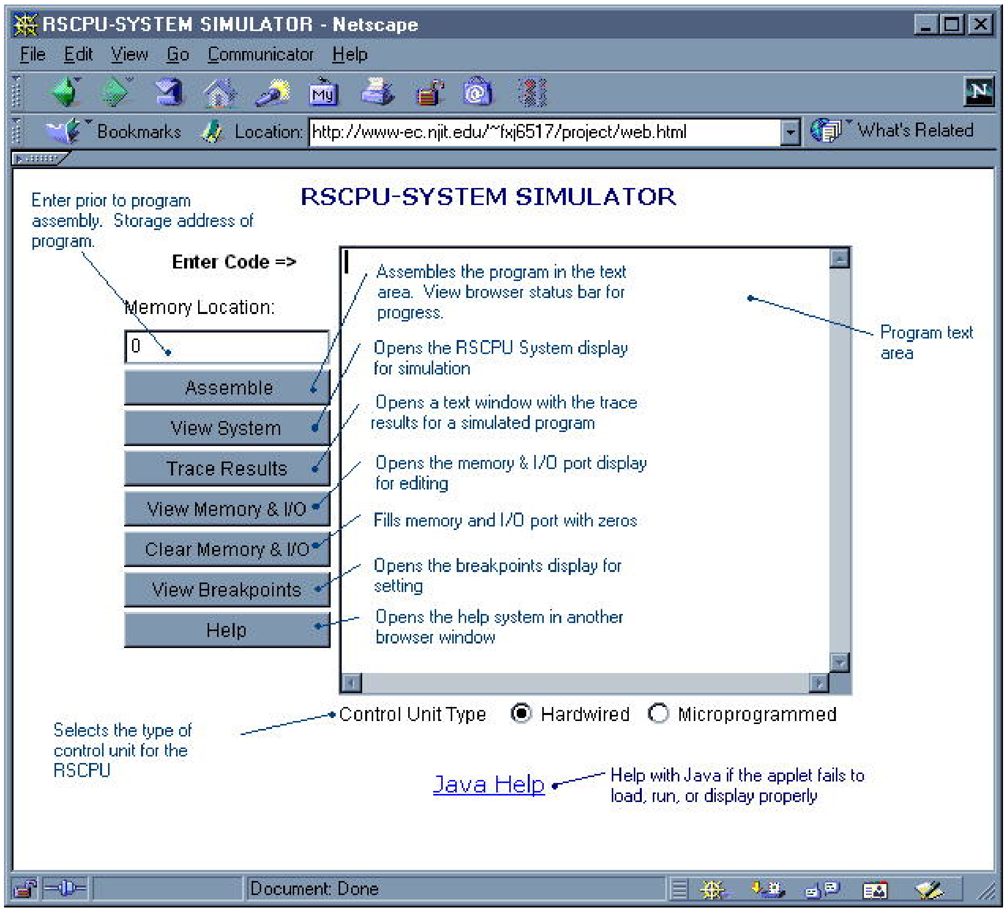
VISUALIZATION TOOL FUNCTIONS

The Relatively Simple Computer System Simulator is a Java applet that runs in any web browser with the Java 2 Virtual Machine version 1.4 plug-in, freely available online.[6] The multimedia quick-start guide requires the Macromedia Flash plug-in.[7]

After starting the simulator, it presents the opening screen, shown with annotations in Figure 2.

Typically, the user starts by entering an assembly language program in the program text area and assembling the program. The simulator lists any errors encountered, and the user corrects and re-assembles the program until it assembles properly. Once this is done, 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; this is useful for entering data to be used by the program. The user may select either a hard-wired or microcoded control unit for the visualization.

The user may simulate and visualize the behavior of the system in one of several modes. The user can execute a program in its entirety, only viewing the contents of the registers and memory after the program has terminated. This is useful when the user is primarily concerned with creating working code rather than visualizing the functions of the system. Also, the user can set breakpoints within the program, continuously executing code until a breakpoint is reached. For more detailed simulation of instructions, the user can also execute individual instructions, as well as step through the operations which occur during individual clock cycles as the instruction is


RSCPU-SYSTEM SIMULATOR - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://www-ec.njit.edu/~fxj6517/project/web.html
What's Related
RSCPU-SYSTEM SIMULATOR
Enter prior to program assembly. Storage address of program.
Enter Code =>
Memory Location:
0
Assemble
View System
Trace Results
View Memory & I/O
Clear Memory & I/O
View Breakpoints
Help
Assembles the program in the text area. View browser status bar for progress.
Opens the RSCPU System display for simulation
Opens a text window with the trace results for a simulated program
Opens the memory & I/O port display for editing
Fills memory and I/O port with zeros
Opens the breakpoints display for setting
Opens the help system in another browser window
Program text area
Selects the type of control unit for the RSCPU
Control Unit Type
Hardwired
Microprogrammed
Java Help
Help with Java if the applet fails to load, run, or display properly
Document: Done

Control Unit Type: [✓ Hardwired] Microprogrammed

Figure 2: Relatively Simple Computer System Simulator opening screen (annotated)

fetched, decoded, and executed. In every mode, the user may trace the execution of some or all of the instructions. Using the trace option, the user may display the contents of all registers or only those registers in the instruction set architecture, either after every instruction or after every clock cycle.

The "View System" button in Figure 2 brings up a separate window with the system display, as shown in Figure 1. This window illustrates the system, as well as the contents of the registers, within the Relatively Simple CPU. The user can view all registers within the CPU, or only the *AC*, *R*, and *Z* registers in its instruction set architecture. The user simulates the execution of the assembly language program using this window.

Regardless of the execution mode, the user starts the system by specifying the memory start address. The simulator then animates the flow of data on the address and data buses using dots that move along the bus in the direction of data flow. The simulator also highlights the signals on the control bus when they are active by displaying them in red. This is done for every clock cycle of every instruction to give students a more intuitive feel for how data moves between components within a computer system as it fetches, decodes, and executes instructions. To facilitate continuous mode execution, the user may disable animation, such as when checking to see whether or not a program produces the correct results.

OTHER SIMULATORS

We have developed several simulators/visualization tools at the New Jersey Institute of Technology for computer architecture education. This paper has described the Relatively Simple Computer System Simulator. This simulator and its source code are available on the companion web site for the textbook on which it is based.[8,9] Other simulators developed through this project are listed below; their source and executable code are also available at these same web sites, and are also freely available under the terms of the GNU General Public License.[10]

RELATIVELY SIMPLE CPU SIMULATOR: This simulator[11] illustrates the flow of data between components within the Relatively Simple CPU as it fetches, decodes, and executes instructions. It also uses animation to illustrate data flow and highlights asserted control signals to give users a more intuitive feel for how the CPU works. As with the Relatively Simple Computer System Simulator, it allows the user to enter and simulate the execution of programs in direct mode, using breakpoints, or by single stepping, either by instruction or by clock cycle.

VERY SIMPLE CPU SIMULATOR: Before presenting the Relatively Simple CPU in[5], an even simpler, 4-instruction CPU is presented and designed. This simulator[12] functions like the Relatively Simple CPU simulator for this simpler CPU.

PROGRAMMABLE LOGIC DEVICE SIMULATOR: This simulator illustrates how a generic, low-density PLD functions. The user selects the number of inputs and outputs, up to four for each, and makes and breaks connections within the PLD to “program” functions into the device.

WALLACE TREE SIMULATOR: Wallace Trees are combinatorial logic circuits used to multiply binary integers. Constructed using Carry-Save Adders, they are a fast, efficient method to implement multiplication. Since these adders do not propagate carry values between bits, they are faster than parallel adders and can produce multiplication products faster than other multiplication hardware. This simulator[13] allows students to multiply binary values with 4, 6, or 8 bits, illustrating the functions of each component within the Wallace tree and how it generates the final product of these values.

ACKNOWLEDGMENTS

Development of the Relatively Simple Computer System Simulator was supported by the National Science Foundation through the Gateway Engineering Education Coalition, and the United States Department of Education via the Ronald E. McNair Postbaccalaureate Achievement Program.

Additional funding for development of the other visualization tools created through this project was provided by the New Jersey Center for Multimedia Research and through the New Jersey Institute of Technology’s Provost's Challenge Grant Program.

REFERENCES

1. Patterson, D. and J. Hennessy. Computer *Organization & Design: The Hardware/Software Interface*, 2nd edition, San Francisco: Morgan Kaufmann Publishers, 1998.

2. Stallings, W. *Computer Organization and Architecture, 5th edition*, Upper Saddle River, NJ: Prentice Hall, 2000.

3. Tanenbaum, *A. Structured Computer Organization, 4th edition,* Upper Saddle River, NJ: Prentice Hall, 1999.

4. Mano, M. *Computer Systems Architecture, 3rd edition,* Upper Saddle River, NJ: Prentice Hall, 1993.

5. Carpinelli, J. *Computer Systems Organization and Architecture*. Reading, MA: Addison-Wesley, 2001.

6. URL: java.sun.com; Sun Microsystems Corporation web site for Java tools and utilities.

7. URL: www.macromedia.com; Macromedia Corporation web site.

8. URL: www.awl.com/carpinelli; Companion web site for *Computer Systems Organization and Architecture*.

9. URL: www.awl.com/info/carpinelli; Companion web site for *Computer Systems Organization and Architecture*.

10. URL: www.gnu.org; GNU General Public License.

11. Carpinelli, J. “The Relatively Simple CPU Simulator,” *ASEE Computers in Education Journal,* vol. XII, no. 2, April-June 2002, pp. 20-26.

12. Carpinelli, J. “The Very Simple CPU Simulator,” *Proceedings of the 2002 Frontiers in Education Conference,* Boston, MA, November 2002.

13. Carpinelli, J. and M. Dokachev. “The Wallace Tree Simulator,” *Proceedings of the 2003 American Society for Engineering Education Conference,* Nashville, TN, June 2003.

BIOGRAPHICAL INFORMATION

John D. Carpinelli is an associate professor of Electrical and Computer Engineering, and Computer and Information Sciences, at New Jersey Institute of Technology. His research interests include interconnection networks, computer architecture, parallel processing, distance learning, and computer simulation. He is the author of the textbook Computer Systems Organization and Architecture (Addison-Wesley, 2001).