A Graphical Computer Simulator for Systems Programming Courses

Mark Newsome and Cherri M. Pancake
Department of Computer Science and Engineering
Auburn University, AL 36849
mnewsome@eng.auburn.edu, pancake@ducvax.auburn.edu

xSICSIM *is an X-based graphical interface for the SICSIM computer simulation tool. Its graphical portrayal of register-level components transforms the "black box" SICSIM machine into a "visual computer", helping students understand how computers work. Single step, fast-execution, and breakpoints are among the control features helpful for debugging assembly language programs. Automatic disassembly and format conversions and displays for comparing expected to actual execution reduce frustration in debugging loader, macro processor, and assembler projects.*

Overview

Computer simulation tools (CSTs) are becoming increasingly popular in the computer science curriculum. These tools employ software simulation to mimic the behavior of other hardware systems. From an educational standpoint, CSTs offer several benefits:

- By using software layers to extend the capabilities of the underlying system, a CST can provide students with hands-on experience, without the need to invest in specialized hardware. Tools such as PARALLAXIS [1], for example, simulate costly SIMD and MIMD multiprocessor systems.
- A CST can monitor the execution of student programs, providing feedback in situations where direct execution would be impossible or impractical. VHDL [2], for example, simulates machine execution based on an architectural specification, allowing the student to test his or her design.
- Because the CST intervenes between the user and the system, it can mask irrelevant details in order to maximize the educational experience. ASSIST [3], for example, hides the intricacies of assembly-language I/O by providing a series of high-level pseudo-instructions.
- Since a CST is implemented completely in software, it can simulate hypothetical as well as real systems. MicMac [4] and CPU SIM [5], for example, allow students to study computer organization through abstract instructions which approximate, but do not duplicate, the behavior of actual machines.
- Because they are designed for use in a variety of educational settings, CSTs often are supported across a range of computer systems. The SICSIM program [6], for example, is written in standard Pascal and has been installed on such diverse platforms as VAX/VMS, MS-DOS, and UNIX.

Unlike computer-assisted instructional tools in other disciplines, however, existing CSTs employ only the most rudimentary user interface mechanisms. They rely on typed commands for input and, with few exceptions, provide only a single text-based output stream (CPU SIM and other Macintosh-based systems make use of multiple windows, but their contents are purely textual). Tutorial facilities are noticeably lacking, and online help is limited to terse explanations of command syntax.

In the systems programming course at Auburn University, we employ the SICSIM simulator, developed by Leland Beck to accompany his popular text [7]. Source code for SICSIM is distributed with the text and is easily installed on most Pascal hosts. It simulates the SIC/XE, a hypothetical machine designed to reflect the features of such diverse systems as IBM 360/370, DEC VAX, and

Intel 80x86. Beck's model is admirable in its ability to expose students to the variety of instruction formats and addressing modes commonly found on popular "real" machines, while ignoring their idiosyncracies. SIC/XE's 59 instructions are divided into four formats which support 18 addressing modes, integer and floating point arithmetic, and device-level I/O. The model provides a uniform basis for teaching the fundamentals of assembler, linker, loader, macro processor, and to some extent, compiler design. The accompanying simulator makes it possible for students to write systems software for the SIC/XE, then execute their "object code" and observe the results.

```
SIC SIMULATOR V1.5
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
S  1
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
D R  2
A=FFFFFF  X=FFFFFF  L=FFFFFF  B=FFFFFF
S=FFFFFF  T=FFFFFF  P=000000  CC=LT
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
D 0000-0090  3
0000 6910087A 1740034B 10002E03 40002900
0010 01332007 4B10005C 3F2FEC03 20590F20
0020 59010003 0F40004B 10005C3E 4003B410
0030 B400B440 75100800 E3203A33 2FFADB20
0040 34A00433 200857A0 31B8503B 2FEA57A0
0050 296D0001 90411340 004F0000 B4107740
0060 00E32012 332FFA53 A010DF20 09B8503B
0070 2FEF4F00 00F10645 4F460000 00000000
0080 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0090 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
R  4
  1000 INSTRUCTIONS EXECUTED
P=00006F
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
H 1  5
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
R  6
  1 INSTRUCTIONS EXECUTED
P=000061
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
R  7
  1 INSTRUCTIONS EXECUTED
P=000064
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
D 0000 - 0090  8
NO ENDING ADDRESS SPECIFIED
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
E RA FF  9
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
E 0005,EA  10
INVALID ADDRESS SPECIFIED
INVALID MEMORY CONTENTS SPECIFIED
COMMAND:S(TART,R(UN,E(NTER,D(UMP,H(COUNT,B(KPT,Q(UIT?
Q  11
```

Figure 1. Sample SICSIM session; numbered lines indicate input typed by the user.

After SICSIM was adopted at Auburn three years ago, it soon became clear that the clumsy interface was frustrating student attempts to employ the tool. Figure 1 presents the output from an interactive SICSIM session. Although there are only five commands, they are cryptic (one letter each), somewhat ambiguous (e.g., *Start* vs. *Run*), and unforgiving (compare the effect of blanks in lines 8 and 3). All information on program execution is in hexadecimal notation, requiring that the student hand-disassemble instructions and manually convert data elements to integer or character form. Error detection is almost non-existent. Students complained that testing and debugging their results was easier by comparing it to printed output than by using the simulator.

To improve SICSIM's usability, we added a graphical user interface. The resulting tool, xSICSIM, utilizes the X Window System [8] platform to provide simple and intuitive access to the basic functionality of the simulator (see Figure 2). xSICSIM's rich visual feedback — in the form of LED-like light displays and color highlighting — helps students see the results of program execution. The graphical presentation also increases student interest, making learning more fun. As the simulator executes instructions, the user can observe changes to the contents of memory, registers, and the program counter. The information can be viewed and manipulated in several familiar formats (hexadecimal, decimal, binary, and ASCII), with automatic conversion between them. An automatic disassembly feature dissects the current instruction into opcode, addressing mode, and operand fields, indicating how the object code is interpreted by the simulator. xSICSIM's interface mechanics, based on single-stroke access to common functions, reduce the likelihood that new errors will be introduced by typing mistakes. Finally, students are more productive because bugs are easier to identify and fix.

This paper describes the structure of xSICSIM. Although the interface clearly was designed to enhance the functionality of Beck's SICSIM, the principles are equally applicable to a variety of related tools (e.g., MicMac [4], CPU SIM [5], MIC-10 [9], SIM68 [10]) used in teaching computer organization, assembly-language programming, and systems programming.

Components of xSICSIM

The xSICSIM system exploits recent advances in graphical interface technology. Its overall structure

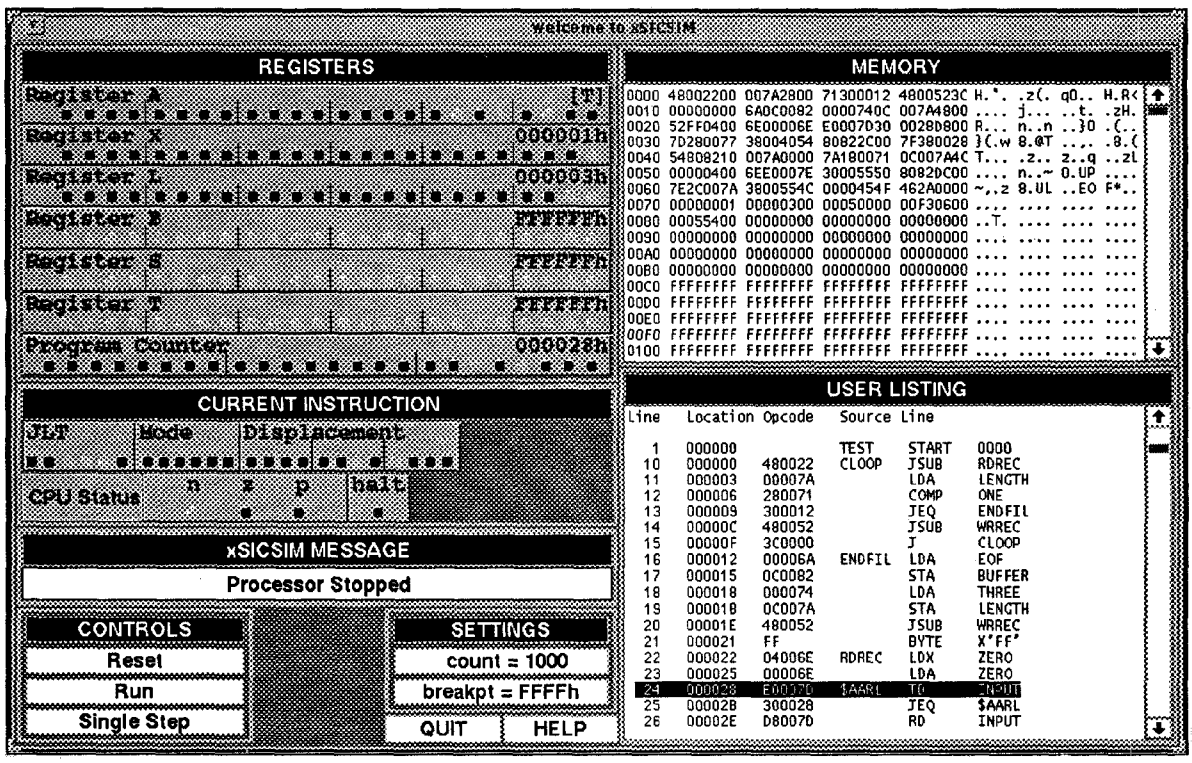

Figure 2. Frame from sample xSICSIM *session.*

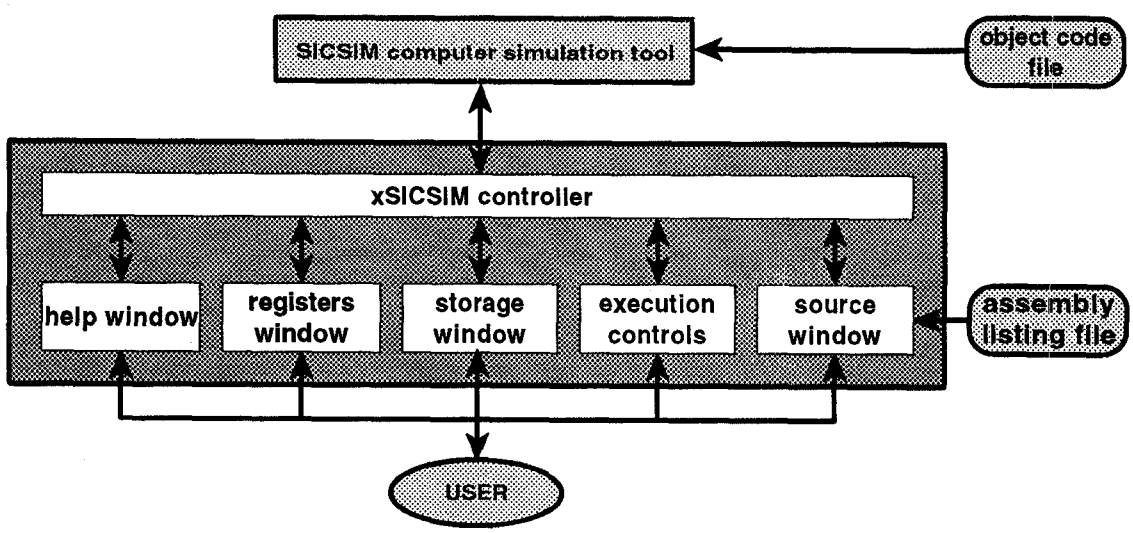

Figure 3. Structure of xSICSIM.

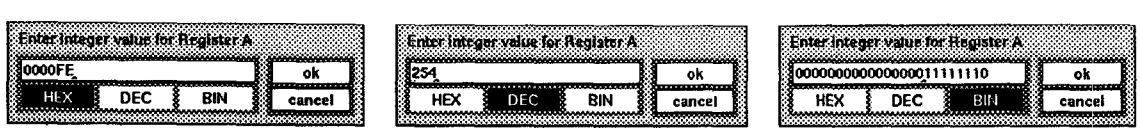

Figure 4. Popup windows control the value and format of each register.

is depicted in Figure 3. Beck's SICSIM simulator provides the basic functionality, but is no longer in direct contact with the user. The `xSICSIM` interface hides the command-driven simulator completely; it runs SICSIM in the background, taking advantage of UNIX multitasking and interprocess communication via pipes.

In the X Window System tradition, the user interacts with interface objects displayed on the screen, which in turn communicate with the `xSICSIM controller`. All user input is captured, interpreted, and responded to by the controller or reformulated as one or more commands to SICSIM. SICSIM output is likewise intercepted and reformatted for display in one of the interface objects.

The interface objects are implemented using standard X Window System widgets from the Athena widget set [11], augmented by specialized widgets designed at Auburn. The nature and use of each object is described below.

Execution Control. `xSICSIM` is operated via pushbutton panels (the white buttons in Figure 2). Selecting `Reset` cold-starts the SIC/XE processor with a "hardware reset." This causes the machine to set all memory locations to a hex value of FF, clear all registers to 00, and reset all I/O devices. `Reset` also precipitates several "housekeeping" tasks that previously had to be handled by the user (such as invoking the SICSIM relocating loader), as well as loading the USER LISTING window with the assembly-language listing file. After a `Reset` operation has been performed, the object code is in memory and awaits execution.

The `Single Step` option allows the user to step through program execution one instruction at a time. Alternatively, `Execute` may be used as a toggle to start/stop rapid execution of the program. Pressing it resumes execution at the location indicated by the Program Counter and changes the button label to `Stop`. A subsequent press halts execution and restores the label to `Execute`. Execution also stops automatically when a HLT (HALT) instruction is executed, an error (illegal instruction, ALU, or I/O error) is detected, or a breakpoint is reached.

Special settings are used to control the rapid-execution options supported by the simulator. The `count` button manages the so-called execution counter, which reflects the number of instructions executed since the last `Reset` operation or count expiration. When the value of the counter has been reached, execution halts and a message is displayed in the MESSAGES area. Selecting `breakpt` activates a popup window where the user indicates breakpoint addresses. Whenever a breakpoint location is accessed during the instruction fetch sequence, execution is halted and a message is issued to the MESSAGES area. Execution may then be resumed by the `Execute` or `Single Step` mechanisms. The user may also start execution at an arbitrary location in the program by selecting the Program Counter register, entering the new location, and and resuming execution via `Single Step` or `Execute`.

Register Displays. The REGISTERS window contains LED panels representing the SIC/XE registers: A (Accumulator), X (Index), L (Linkage), B (Base), S (General Purpose), T (General Purpose), and Program Counter, each 3 bytes in length.[1] Integers are stored as 24-bit binary numbers, using 2's complement to represent negative values, while characters make use of standard 8-bit ASCII codes.

The LEDs display the binary representation of the register values, each light representing one bit. In addition, the hexadecimal (or optionally, the ASCII) value appears in the upper lefthand corner of each register display; the display mode is changed by clicking the righthand mouse button while the cursor is positioned over the register. A register's value may be changed via a popup window, activated when the lefthand mouse button is pressed. The popup also allows the student to view the register contents in binary, hexadecimal, or decimal format (see Figure 4).

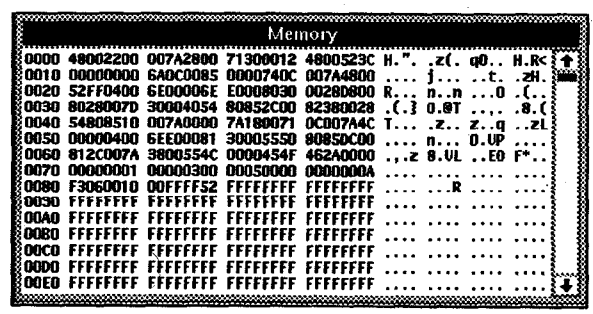

Figure 5. The MEMORY DUMP window.

[1] Although Beck's model also supports a 6-byte floating-point register, its is not supported by the SICSIM simulator and therefore cannot be used in student programs.

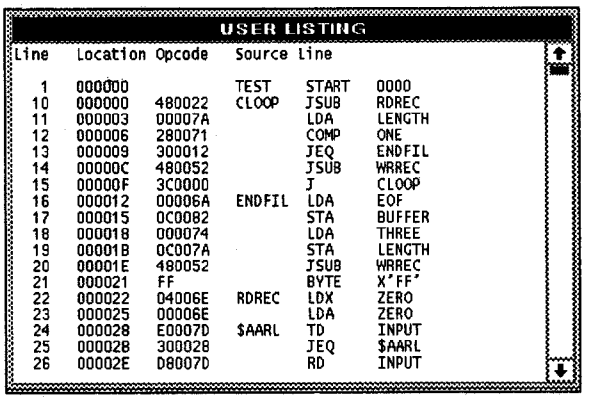

Figure 6. The USER LISTING window.

Storage Display. The scrollable MEMORY DUMP window (Figure 5) displays the contents of memory in both hexadecimal and ASCII representations. To alter storage values, the user presses the lefthand mouse button while the cursor is positioned over the window. This activates a data entry popup similar to the register popup. Again, values may may be entered in any format. The MEMORY scrollbar is used to bring the desired region of memory into view. Clicking on the arrows located at either end of the bar causes the display to scroll one line; alternatively, the thumb (grey region inside the bar) can be dragged until the desired area is visible.

Source Code Display. The program is executed directly from the student's object code file, and reflects any anomalies in its format. To facilitate the identification of incorrect object instructions, the USER LISTING window (Figure 6) provides quick access to the "expected" instructions — the original assembly-language instruction, the location counter, and the object code generated by the assembler. As in the MEMORY window, a scrollbar allows easy navigation through the program text.

Help and Tutorial Features. On-line help is always available via popup windows (see Figure 7). The hierarchically-organized help system makes use of `xhelp` (an X utility also developed by the authors) to provide a tutorial-style introduction to the mechanics of the interface, the SIC/XE machine architecture, SIC/XE assembly language, and details of program execution.

Selecting the master help button, labeled HELP, enters the help system at the top level and is used primarily by newcomers to `xSICSIM`. In addition, the help facility is activated in context-sensive mode whenever the user clicks on a window titlebar (shown here as white text on a black ground); appropriate information about the window is displayed in response. For example, clicking on the REGISTERS titlebar provides information on the number and purpose of SIC/XE registers, as well as how the display objects can be manipulated to change format, register contents, etc.

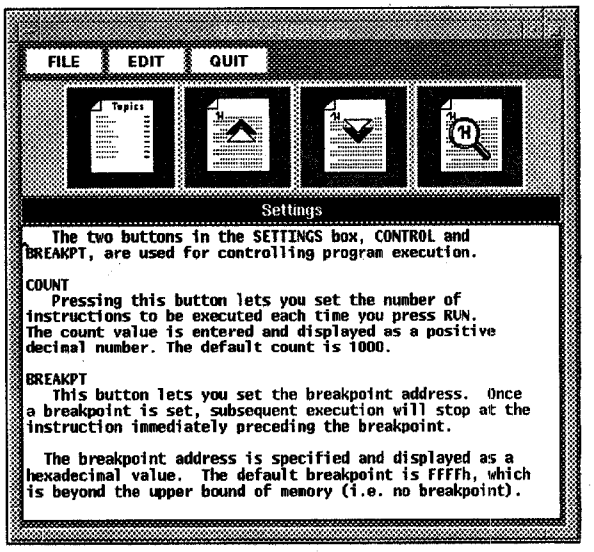

Figure 7. The popup HELP facility.

Experiences with xSICSIM

Written in a highly portable subset of the C language, `xSICSIM` can be installed on any system supporting release X11R4 or later of the X Window System. To date, it has been ported to UNIX workstations (Sun 3/50, SparcStation, and IPC), IBM/AIX systems (RISC System 6000 and X Stations). With the exception of one minor adjustment,[2] the interface preserves the total independence of SICSIM, which may still be invoked via its command-line interface.

`xSICSIM` was used successfully over the past year in CSE400L, the systems programming laboratory course required for undergraduates in the Computer Science and the Computer Engineering curricula at Auburn. The response has been very favorable. In particular, the graphical interface is cited as being both appealing and useful.

[2] SICSIM is written in Pascal, notorious for its I/O buffering. The program was modified to flush the buffer after each write, ensuring that `xSICSIM` receives an immediate response from the simulator; otherwise, unacceptable delays would occur in the display as the buffer filled.

Judging from the quality of student implementations of assemblers and macro processors, xSICSIM increases student interest and improves their understanding of the interrelationships between source code, object code, and CPU operation. Students report that the tool's error detection capabilities — which identify illegal opcodes, malformed instructions, and out-of-range addresses — are helpful in localizing mistakes or misalignments in the object code format, a task which previously was frustrating and time-consuming. In effect, xSICSIM converts the "black-box" SICSIM machine into a "visual computer" which reinforces the student's mental model of computer operation. By observing how the machine interprets and responds to each instruction, he or she can identify the cause and effect of object code anomalies, without the tedium of manual format conversions and disassembly.

References

[1] Barth, Ingo, Thomas Braunl, and Frank Sembach, *Parallaxis User Manual*, Technical Report 3/90, Universitaet Stuttgart, Computer Science Department (March 1990).

[2] Coelho, David R., *The VHDL Handbook.* Boston, Kluwer Academic Publishers (1989).

[3] Overbeek, Ross A., *Assembler langauge with ASSIST and ASSIST/I*, Chicago, Science Research Associates (1986).

[4] Donaldson, John L., "MicMac: a Microprogram Simulator for Courses in Computer Organization," *ACM SIGCSE Bulletin*, 20 (3): 428–431 (September 1988).

[5] Skrien, Dale and John Hosack, "Multilevel Simulator at the Register Transfer Level for Use in an Introductory Machine Organization Class," *ACM SIGCSE Bulletin*, 23 (1): 347–351 (March 1991).

[6] Beck, Leland L., "SICSIM documentation," unpublished manuscript from San Diego State University, distributed by Addison-Wesley (1990).

[7] Beck, Leland L., *Systems Software: An Introduction to Systems Programming*, Reading MA, Addison-Wesley (1990).

[8] Scheifler, Robert and James Gettys, *X Window System: Complete Reference to Xlib, X Protocol, ICCCM, XLFD*, Second Edition, Bedford MA, Digital Press (1990).

[9] Sayers, Jerry E. and David E. Martin, "A Hypothetical Computer to Simulate Microprogramming and Conventional Machine Language," *ACM SIGCSE Bulletin*, 20 (4): 43–49 (December 1988).

[10] Allen, Laura "Sim68000, A Simulator for the MC68000 (Version 2)," unpublished report distributed by University of Florida (1990).

[11] Peterson, Chris D., *Athena Widget Set - C Language Interface*, Cambridge MA, Massachusetts Institute of Technology and Digital Equipment Corporation, 1989.