MARS: An Education-Oriented MIPS Assembly Language Simulator

Dr. Kenneth Vollmar
Missouri State University
Springfield, MO 65897
417-836-5789
KenVollmar@missouristate.edu

Dr. Pete Sanderson
Otterbein College
Westerville, OH 43081
614-823-1317
PSanderson@otterbein.edu

ABSTRACT
We describe the implementation of "MARS," a GUI, Java-based simulator for the MIPS assembly language. MIPS, the computer architecture underlying the simulated assembly language, is widely used in industry and is the basis of the popular textbook *Computer Organization and Design* [6], used at over 400 universities. The MARS simulator has been implemented with characteristics that are especially useful to undergraduate computer science students and their instructors.

Categories and Subject Descriptors
C.0 [**Computer Systems Organization**]: General – *Hardware/software interfaces, Instruction set design (RISC), Modeling of computer architecture.*

General Terms
Languages

Keywords
Architecture, assembly language, simulation, MIPS

1. INTRODUCTION
The MIPS RISC architecture and corresponding assembly language use a limited number of instruction formats. Typical student programs may use register-to-register, load/store, branch, jump, system call, and floating-point instructions. Thirty-two general-purpose registers are available for integer operations (some have dedicated uses), as are thirty-two single-precision floating point registers. MIPS is a clean design with simple instructions, and is very popular in industry as well as academia.

The widely-used *Computer Organization and Design* [6] text is based on the MIPS architecture and instruction set. Since computer science and computer engineering departments may not have adequate access to MIPS equipment to support laboratory activities, software-based MIPS simulators may be used. Additional reasons for using simulation software in an organization and architecture course are described in [9], and two issues of the ACM Journal of Educational Resources in Computing (JERiC) were devoted to computer architecture simulators for educational purposes [10][11]. The SPIM [5] simulator is bundled with *Computer Organization and Design* and described in its Appendix.

Our goal for this project was to create an alternative to SPIM specifically for the needs of typical undergraduate students and their instructors. It should be useful in courses such as computer organization and architecture, assembly language programming, and compiler writing. The resulting simulator is called MARS (**M**IPS **A**ssembler and **R**untime **S**imulator) [8]. MARS is an Integrated Development Environment (IDE) controlled by a modern GUI whose features include:

- control of execution speed, including single step at variable speed (slider bar controls the number of instructions per second)
- thirty-two registers visible at the same time, selectable via tabbed interfaces,
- "spreadsheet" (WYSIWYG) modification of values in registers and memory,
- selection of data value display in decimal or hexadecimal,
- resizable windows,
- "surfing" through memory using buttons to change display to next/previous, stack location, global partition, and the start of the memory segment,
- toolbar icons for every menu item
- an integrated editor and assembler as part of its IDE.

The MARS simulator implements the educationally important portions of the MIPS instruction set utilized by *Computer Organization and Design Third Edition* (COD3) [6]. Specifically, the MARS simulator implements:

- All the instructions in the left-hand column of COD3 Figure 3.24, p. 226, which are the primary concentration of the text.
- All the pseudo-instructions in the right-hand column of COD3 Figure 3.25, p. 227.
- All of the instructions in the right-hand column of COD3 Figure 3.24, p. 226 (MIPS arithmetic core) and left-hand column of COD3 Figure 3.25, p. 227 (remaining MIPS-32).
- The seventeen syscalls in COD3 App. A-48, including file open, read, write, and close.

SIGCSE'06, March 1-5, 2006, Houston, Texas, USA.

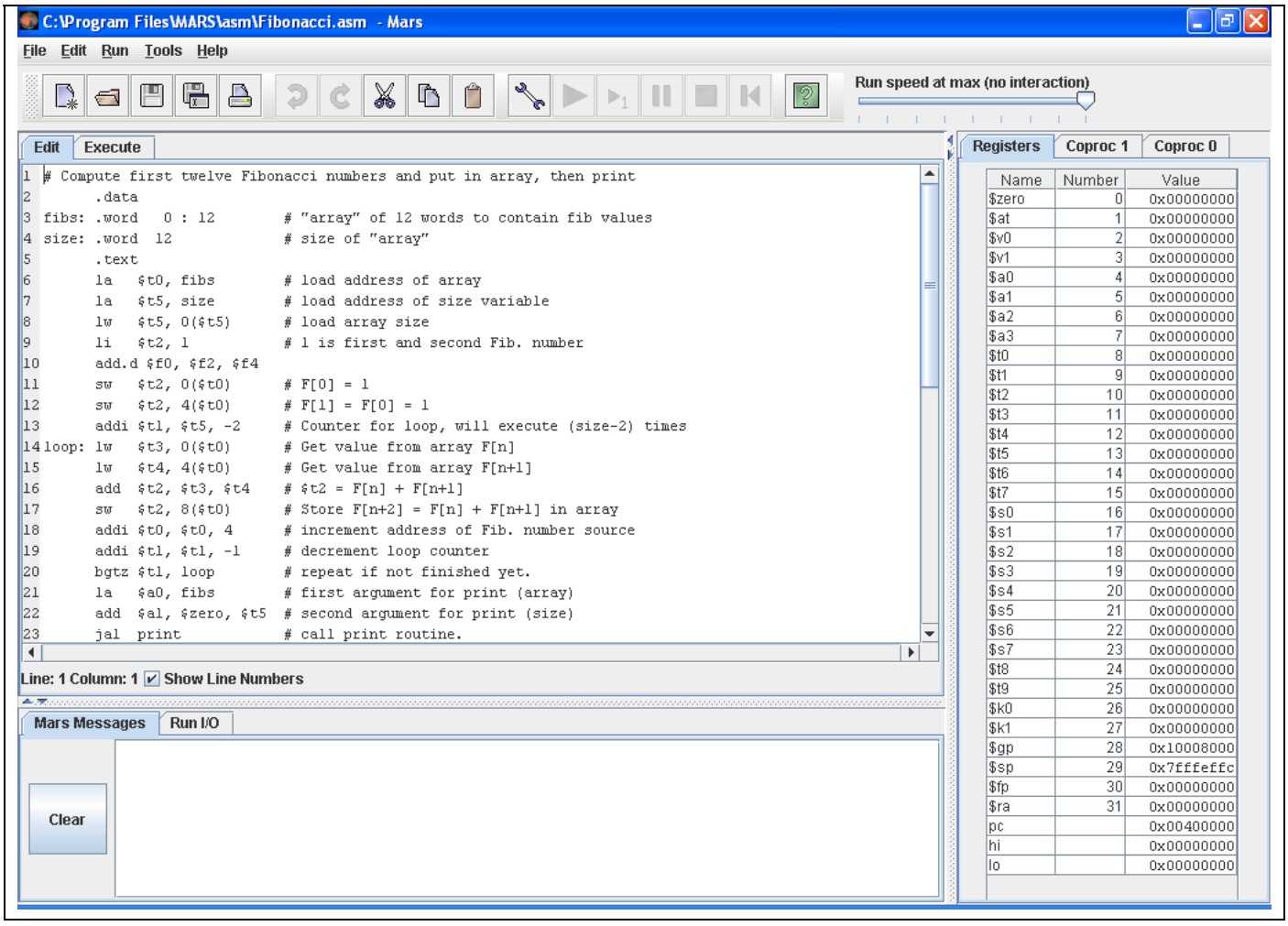

Figure 1. MARS editor window is active ("Edit" tab is foremost).

Instructions not yet implemented in MARS include some pseudo-instructions and other instructions expected to be more of professional than educational interest.

The MARS simulator operates under either GUI or command-line modes of operation. Students use primarily the GUI mode in either "go" or "single step" execution for assembly code creation and debugging. Instructors have the option of running the simulator from an OS shell or a batch command file, to facilitate execution of several test cases of all student's programs in sequence for grading. Command-line arguments are used to request the output of particular registers or memory locations to verify program results.

The MARS simulator is written in Java 1.4.2, using standard techniques of human-computer interaction via its Swing and AWT packages. Standard icons have been obtained from the Java look and feel Graphics Repository [7].

2. MARS OPERATION

Students will typically use MARS to compose an assembly language program using the editor, assemble it, then execute the assembled program all at once or step-by-step using the facilities of the execute pane. These operations are illustrated and described below.

The MARS editor is an ASCII-oriented text editor that operates much like Window's Notepad. Figure 1 shows the active editing pane. The first two groups of toolbar icons are used with the editor. The first group corresponds to the *File* menu and includes file options such as *New*, *Open*, and *Save*. The second group corresponds to the *Edit* menu and includes operations such as *Cut*, *Copy* and *Paste*. Menu items and their corresponding toolbar buttons are enabled and disabled as appropriate.

To assemble the program, the user selects *Assemble* from the *Run* menu or clicks the wrench toolbar icon. A successful assembly causes the Execute pane to come forward as shown in Figure 2. An unsuccessful assembly displays appropriate messages and line numbers in the console window at the bottom of the screen.

The Execute pane contains several windows. The Text Segment window is front and center. It displays both the source and binary code of the assembly program, including the expansion of pseudo-instructions (the `la` and `li` instructions in Figure 2). A breakpoint can be set at any instruction using the check box in the

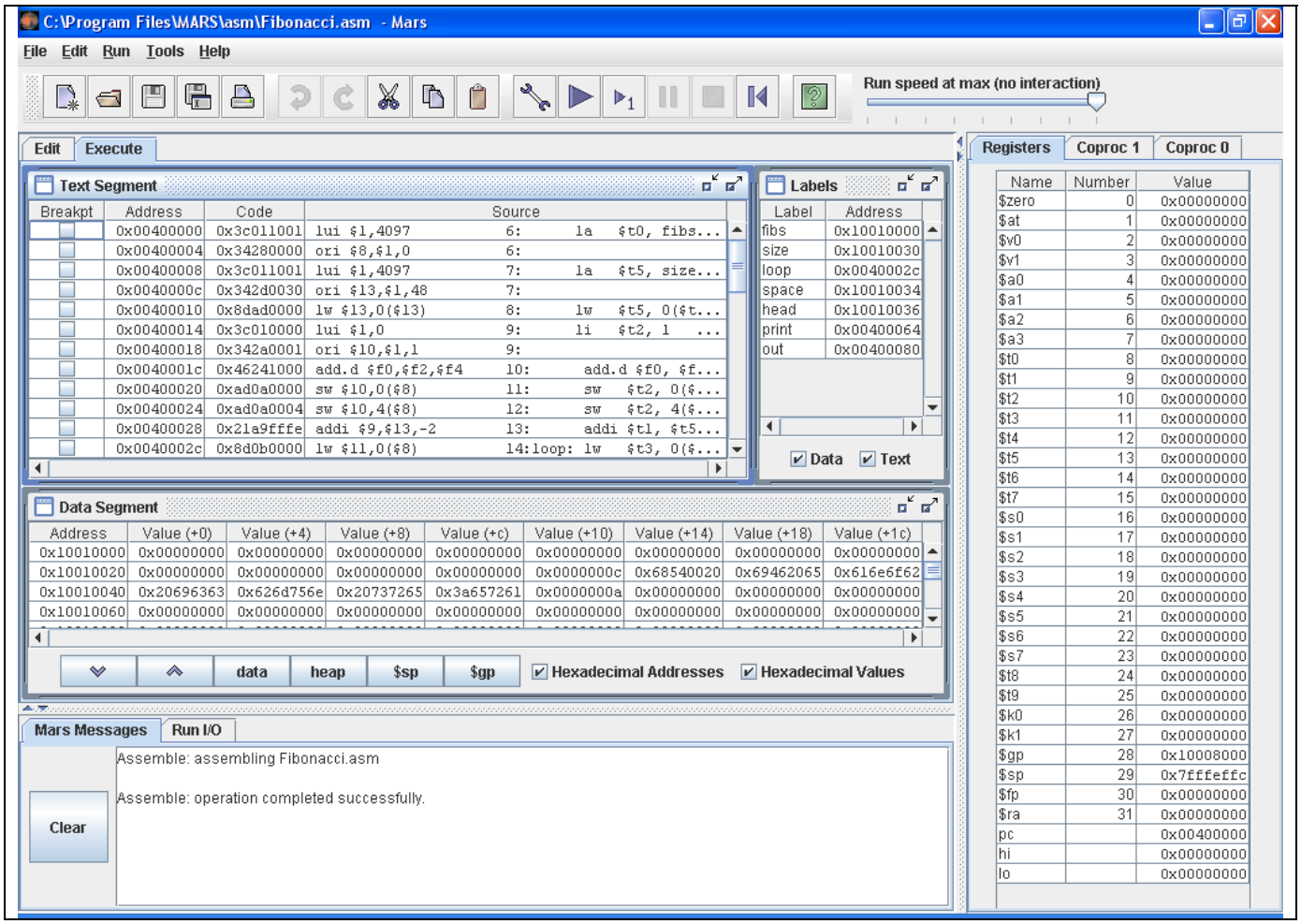

Figure 2. MARS execution window is active ("Execute" tab is foremost and the execution toolbar icons are active).

leftmost column. When stepping through program execution manually or at reduced run speeds, the next instruction to be executed is highlighted.

The Data Segment display illustrated at the bottom of Figure 2 shows the program's data storage area in a scrollable window. Its lower border contains icons to control display of memory contents at special locations such as the stack or heap, and check boxes to display memory addresses and values in either decimal or hexadecimal format. The contents of a memory word can be directly edited at any time by double-clicking on its cell and entering the desired value in either decimal or hexadecimal format.

Symbol table information is displayed in the Labels window. This is relatively less important and the window may be closed to allow more space for the Text Segment display.

Registers are permanently displayed to the right of the Execute pane in a vertically oriented window. This can be seen in the right side of Figure 1. As with memory, values are editable and display format is selectable. There are separate tabs for the general purpose registers, the floating point registers of Coprocessor 1 and the exception registers of Coprocessor 0.

Another permanent display is the console window on the lower portion of the screen. It includes two tabs, one for MARS messages such as assembly errors and another for runtime input and output generated by MIPS system calls. Each tab is activated when text is written to it.

3. SPIM AND OTHER MIPS SIMULATORS

A number of MIPS simulators have been developed over the years. Most can be classified by a small number of categories: those designed for research use (e.g. MIPSI), those that focus on certain MIPS architectural features such as pipelines (e.g. WebMIPS [2], SmallMIPS, RTLSim), those that depend on SPIM (e.g. MIPSASM, TinyMIPS), and general purpose simulators. Examples of the latter include MipsIt [3] and SPIM [5]. Most MIPS simulators include features for visualizing and/or animating MIPS components. MARS and SPIM do not.

The COD3 textbook and companion website refer to the SPIM simulator, which is available on its bundled CDROM or from the web. SPIM is without doubt the most widely known and used MIPS simulator, serving both education and industry. MARS has been designed as an alternative to SPIM to meet the needs of

typical undergraduate courses. A comparison of some education-oriented characteristics of SPIM 7.1 to MARS 2.0 follows.

- The SPIM user interface has one window split into five scrollable but non-resizable panes. Using PCSPIM on a 19" monitor, at most nine lines of source code are visible at a time. MARS uses resizable windows and tabbed panes to more easily focus on memory, register or program contents.
- Several steps are required to modify register or memory values in SPIM: calling up a pop-up window, typing the register or address, and specifying the new value. This is time-consuming and error-prone. MARS features WYSIWYG on-the-spot modification.
- Similarly, SPIM's breakpoints are set by calling up a pop-up window and typing the breakpoint location. MARS features a check box beside each line of code to set and remove breakpoints immediately.
- SPIM permits simulated execution to proceed in "run", "single step" or "multiple step" mode. MARS permits the first two, plus offers a variable-speed timed mode (up to 30 instructions per second) with interactive display update, speed adjustment, and WYSIWYG value modification. See slider in upper right of Figure 1. When set to maximum speed, there is no interaction until the program terminates or the Pause or Stop button is selected.
- SPIM does not include an integrated editor, so files must be edited by an external application. You may however easily re-load such a file. MARS includes a basic text editor.

In summary, support for interactive testing and debugging is one of MARS greatest strengths.

The MARS text editor currently provides Notepad-like functionality. Some contextual help is provided by tool tips that appear when the mouse is hovered over the always-present Register window. Thus MARS provides limited support during the program composition phase. This need can be addressed by commercial editors such as Downcast Systems' MIPSter [4].

4. OTHER EDUCATION-ORIENTED FEATURES AND DETAILS

We have largely achieved our original MARS project goal, which was to develop a viable alternative to SPIM for typical undergraduate use. In other words, to provide a tool that implements the important MIPS instructions (those covered in COD3) through a portable and student-friendly IDE.

Along the way, we realized that through MARS we could and should achieve more significant contributions to assembly language and computer organization/architecture education. Here we introduce two aspects of MARS implementation that may represent its larger contribution: external instruction set specification, and tool plug-in capability. Both are partially achieved at this time.

The simplicity and regularity of the MIPS instructions permit the separation of the specification of MIPS instructions from MARS source code. The specification for each instruction consists of:

- an example usage of the instruction
- the instruction format
- a template of the generated 32 bit machine instruction with operand positions indicated
- a Java method to simulate the execution of the instruction

All except the last are strings that may be placed in a textual configuration file for loading when MARS is launched. Similarly, a separate text file is used to specify MIPS "pseudo-instructions" (a.k.a. macro instructions). Pseudo-instructions are expanded into one or more native MIPS instructions by the assembler. For each pseudo-instruction, the text file contains a specification consisting of an example usage followed by a tab-separated list of native instructions into which it will be translated with appropriate operand substitution.

These implementation features could be utilized by instructors and students to design and implement a customized MIPS-like instruction set, then use MARS to assemble and simulate programs written in the new language. Custom simulated native and pseudo instructions are defined by adding the properly formatted specifications into the configuration. Instructors in compiler writing courses may also use this capability to define and implement a simplified target assembly language for student compilers.

Because this capability was not part of the original design, the assembler's tokenizer is not table-driven and thus any customized instruction set would have to follow MIPS lexical formats. This is an issue we would like to address in the future. Nonetheless, we are very excited by the possibilities this capability presents in the classroom.

The tool plug-in capability permits the definition of customized bots, animations, or any number of other useful tools to be controlled by a MIPS program during MARS simulation. A tool "observes" MARS memory locations and reacts appropriately in response to data changes in the memory-mapped IO locations defined for this tool. The source code of a tool is separate from the source code of MARS.

Using a dynamic class-loading technique from game programming [1], any externally-compiled class which implements a certain Java interface and resides in the tools folder will be detected and loaded at MARS launch and added to its Tools menu (see Figure 1). User selection of that Tools menu item will invoke a particular interface method, which will typically establish itself as an **`Observer`** of MARS memory locations. A MIPS program will read and write memory locations and the tool will respond accordingly.

For instance, a "scrolling marquee" tool could graphically simulate a rectangular array of LEDs in which an address represents a vertical column of eight LEDs and an 8-bit data value represents the on/off values of each of those LEDs. A MIPS program could implement a scrolling marquee on the LED array by writing to two memory locations to specify LED address and value. The tool has the responsibility of graphically displaying the LEDs, including the accurate modeling of LED illumination and persistence. The MIPS program has the responsibility of determining the 8-bit values needed for the alphanumeric data, and refreshing the data in the memory-mapped IO locations at the proper rate for the scrolling motion.

We have used this technique to implement a MARSBot similar to [12] for display of the motions of a simulated robot. Interfacing

with physical hardware is possible using a tool which writes and reads an external port of the MARS host computer. One of us has proposed a sabbatical project to develop a virtual world using this capability.

5. STUDENT EVALUATIONS OF MARS

Our computer organization students have used MARS for two semesters now. To rigorously assess our project goal of MARS as a viable alternative to SPIM would require the same group of students to be introduced to both simulators simultaneously and equally – an unrealistic and ineffective use of class time. We attempted to gauge student's preferences between the MARS and SPIM simulators by asking them to complete an anonymous, non-graded comparison of the use of SPIM 7.1 and MARS on one of their own programs from earlier in the semester. We disclosed our own involvement with MARS and asked the students to objectively compare the two.

In the survey that followed, students strongly preferred MARS to SPIM, citing most often user-interface aspects such as breakpoints and icon control. The students' description of simulator characteristics which contribute to the learning of an assembly language seemed to focus on single-step execution and convenient display of information pertinent to the current instruction. Most of the aspects of MARS for which improvements were requested have been implemented in MARS 2.0, including register display layout, floating-point instructions, and hot key control of common operations.

6. AVAILABILITY AND FUTURE PLANS

We have ambitious plans for expanding MARS over the coming year. As of this writing, MARS implements 98 MIPS32 native instructions, 36 pseudo-instructions, and the 17 system calls described in COD3 Appendix A. We plan to continue implementing the remaining instruction set. Other plans include improving debugging support through such features as highlighting of memory/register contents modified in step-by-step execution, and the ability to undo execution steps.

As mentioned above, we plan to develop a virtual world that can be controlled by an executing MIPS program, although the details have not yet been worked out. We would like to improve support for program composition through syntax highlighting and autocompletion a la MIPSter but this is a lower priority. A number of other features may be implemented or improved as time and resources permit.

The MARS jar file is available for downloading at http://www.cs.missouristate.edu/~vollmar/MARS/.

7. REFERENCES

[1] Brackeen, David, Barker, Bret, and Vanhelswue, Laurence, "Developing Games in Java". New Riders Publishing, 2003.

[2] Branovic, I., Giorgi, R. and Martinelli, E., WebMIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education, *Workshop on Computer Architecture Education*, 31st *International Symposium on Computer Architecture*, Munich, Germany, 2004.

[3] Brorsson, M., MipsIt - A Simulation and Development Environment Using Animation for Computer Architecture Education, *Workshop on Computer Architecture Education*, *29th International Symposium on Computer Architecture*, Anchorage AK, 2002.

[4] Downcast Systems, MIPSter 2.0, http://www.downcastsystems.com/mipster/, retrieved 21 November 2005.

[5] Larus, J., SPIM: A MIPS32 simulator, http://www.cs.wisc.edu/~larus/spim.html, retrieved 21 November 2005.

[6] Patterson, D., and Hennessy, J., *Computer Organization and Design: The Hardware/Software Interface, 3rd edition*, San Francisco, CA: Morgan Kaufmann, 2004.

[7] Sun Microsystems, Java look and feel Graphics Repository, http://java.sun.com/developer/techDocs/hi/repository/, retrieved 21 November 2005.

[8] Vollmar, K. and Sanderson, P., A MIPS Assembly Language Simulator Designed For Education. *The Journal of Computing Sciences in Colleges*, Vol. 21, No. 1, 2005.

[9] Wolffe, G., Yurcik, W., Osborne, H. and Holliday, M., Teaching Computer Organization/Architecture With Limited Resources Using Simulators, *ACM SIGCSE Bulletin 34*, (1), 176 - 180, 2002.

[10] Yurcik, W. (guest editor), *ACM Journal on Educational Resources in Computing*, Vol. 1, No. 4, December 2001.

[11] Yurcik, W. (guest editor), *ACM Journal on Educational Resources in Computing*, Vol. 2, No. 1, March 2002.

[12] Zilles, C., SPIMbot: an engaging, problem-based approach to teaching assembly language programming. *ACM SIGCSE Bulletin 37*, (1), 106 - 110, 2005.