**MARS - Mips Assembly and Runtime Simulator**

**Release 4.5**

**August 2014**

**Cool Capability: Plug-in Tools**

Beginning with Release 2.0, MARS is capable of running externally-developed software that interacts with an executing MIPS program and MIPS system resources. The requirements for such a program are:

1. It implements the mars.tools.MarsTool interface.
2. It is part of the mars.tools package.
3. It compiles cleanly into a ".class" file, stored in the mars/tools directory.

MARS will detect all qualifying tools upon startup and include them in its Tools menu. When a tool's menu item is selected, an instance of it will be created using its no-argument constructor and its action() method will be invoked. If no qualifying tools are found at MARS startup, the Tools menu will not appear.

To use such a tool, load and assemble a MIPS program of interest then select the desired tool from the Tools menu. The tool's window will open and depending on how it is written it will either need to be "connected" to the MIPS program by clicking a button or will already be connected. Run the MIPS program as you normally would, to initiate tool interaction with the executing program.

Beginning with Release 3.2, the abstract class mars.tools.AbstractMarsToolAndApplication is included in the MARS distribution to provide a substantial framework for implementing your own MARS Tool. A subclass that extends it by implementing at least its two abstract methods can be run not only from the Tools menu but also as a free-standing application that uses the MARS assembler and simulator in the background.

Several Tools developed by subclassing AbstractMarsToolAndApplication are included with MARS: an Introduction to Tools, a Data Cache Simulator, a Memory Reference Visualizer, and a Floating Point tool. The last one is quite useful even when not connected to a MIPS program because it displays binary, hexadecimal and decimal representations for a 32 bit floating point value; when any of them is modified the other two are updated as well.

Release 3.5 includes new tools, most notably a keyboard and display simulator that allows you to perform memory-mapped I/O (MMIO) using polled and interrupt-driven techniques as described in various references. Click its Help button for more details.

If you wish to develop your own MARS Tool, you will first need to extract the MARS distribution from its JAR file if you have not already done so. All MARS tools must be stored in the mars/tools directory.

Follow the Tutorial Materials link on the MARS homepage to find a tutorial that covers development of MARS Tools.

**Cool Capability: Extending the syscall set or reassigning syscall numbers**

Beginning with Release 3.1, system calls (syscall instruction) are implemented using a technique similar to that for tools. This permits anyone to add a new syscall by defining a new class that meets these requirements:

1. It implements the mars.mips.instructions.syscalls.Syscall interface, or extends the mars.mips.instructions.syscalls.AbstractSyscall class (which provides default implementations of everything except the simulate() method).
2. It is part of the mars.mips.instructions.syscalls package.
3. It compiles cleanly into a ".class" file, stored in the mars/mips/instructions/syscalls directory.

MARS will detect all qualifying syscall classes upon startup and the runtime simulator will invoke them when the syscall instruction is simulated and register $v0 contains the corresponding integer service number.

Syscalls and syscall number assignments in MARs match those of SPIM for syscalls 1 through 17. However if you wish to change syscall number assignments, you may do so by editing the Syscall.properties file included in the release (this requires extraction from the JAR file).

Follow the Tutorial Materials link on the MARS homepage to find a tutorial that covers development of system calls.

**Cool Capability: Extending the instruction set**

You can add customized pseudo-instructions to the MIPS instruction set by editing the PseudoOps.txt file included in the MARS distribution. Instruction specification formats are explained in the file itself. The specification of a pseudo-instruction is one line long. It consists of an example of the instruction, constructed using available instruction specification symbols, followed by a tab-separated list of the basic MIPS instructions it will expand to. Each is an instruction template constructed using instruction specification symbols combined with special template specification symbols. The latter permit substitution at program assembly time of operands from the user's program into the expanded pseudo-instruction.

PseudoOps.txt is read and processed at MARS startup, and error messages will be produced if a specification is not correctly formatted. Note that if you wish to edit it you first have to extract it from the JAR file.

Follow the Tutorial Materials link on the MARS homepage to find a tutorial that covers modification of the pseudo-instruction set.

This document is available for printing on the MARS home page **http://www.cs.missouristate.edu/MARS/**.