MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Introduction

MARS, the **M**ips **A**ssembly and **R**untime **S**imulator, will assemble and simulate the execution of MIPS assembly language programs. It can be used either from a command line or through its integrated development environment (IDE). MARS is written in Java and requires at least Release 1.5 of the J2SE Java Runtime Environment (JRE) to work. It is distributed as an executable JAR file. The MARS home page is **`http://www.cs.missouristate.edu/MARS/`**. This document is available for printing there.

As of Release 4.0, MARS assembles and simulates 155 basic instructions of the MIPS-32 instruction set, approximately 370 pseudo-instructions or instruction variations, the 17 syscall functions mainly for console and file I/O defined by SPIM, and an additional 22 syscalls for other uses such as MIDI output, random number generation and more. These are listed in separate help tabs. It supports seven different memory addressing modes for load and store instructions: `label`, `immed`, `label+immed`, `($reg)`, `label($reg)`, `immed($reg)`, and `label+immed($reg)`, where `immed` is an integer up to 32 bits. A setting is available to disallow use of pseudo-instructions and extended instruction formats and memory addressing modes.

Our guiding reference in implementing the instruction set has been *Computer Organization and Design, Fourth Edition* by Patterson and Hennessy, Elsevier - Morgan Kaufmann, 2009. It summarizes the MIPS-32 instruction set and pseudo-instructions in Figures 3.24 and 3.25 on pages 279-281, with details provided in the text and in Appendix B. MARS Releases 3.2 and above implement all the instructions in Appendix B and those figures except the delay branches from the left column of Figure 3.25. It also implements all the system services (syscalls) and assembler directives documented in Appendix B.

The MARS IDE provides program editing and assembling but its real strength is its support for interactive debugging. The programmer can easily set and remove execution breakpoints or step through execution forward or backward (undo) while viewing and directly editing register and memory contents.

Questions and Comments

Send MARS questions and comments to Dr. Pete Sanderson at `PSanderson@otterbein.edu` or Dr. Ken Vollmar at `KenVollmar@missouristate.edu`. We will respond as quickly as we can but as teaching professors do not have as much time to work on this project as we would like during the school year. We presented papers on MARS at the 2005 CCSC:MW conference and the 2006 SIGCSE Technical Symposium. We presented a tutorial session on MARS at the 2007 CCSC:CP conference and the Tutorial handout is available from the MARS homepage.

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

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Configuration Settings

Releases 3.0 and later include a Settings menu. The Editor and Exception Handler items launch a dialog but the rest are each controlled by a checkbox for selecting or deselecting it (checked means true, unchecked means false). Settings and their default values are:

1. **Display the Labels window in the Execute tab.** Default value is **false**. If selected, the Labels window, which shows the name and associated address for each label defined in the program, will be displayed to the right of the Text Segment.
2. **Provide program arguments to the MIPS program.** Default value is **false**. New in Release 3.5. If selected, a text field will appear at the top of the Text Segment Display. Any argument values in this text field at the time of program execution will be stored in MIPS memory prior to execution. The argument count (argc) will be placed in register $a0, and the address of an array of null-terminated strings containing the arguments (argv) will be placed in register $a1. These values are also available on the runtime stack ($sp).
3. **Popup Dialog for input syscalls (5,6,7,8,12).** New in Release 4.0. Default value is **false**. If selected, runtime console input will be entered using popup dialogs (this was the only option prior to Release 4.0). Otherwise, input is entered directly into the Run I/O tab at the bottom of the screen.
4. **Display memory addresses in hexadecimal.** Default value is **true**. If deselected, addresses will be displayed in decimal. This setting can also be toggled in a checkbox on the lower border of the Data Segment Window.
5. **Display memory and register contents in hexadecimal.** Default value is **true**. If deselected, vlaues will be displayed in decimal. This setting can also be toggled in a checkbox on the lower border of the Data Segment Window.
6. **Assemble a file automatically as soon as it is opened,** and initialize the File Open dialog with the most-recently opened file. Default value is **false**. This is convenient if you use an external editor for composing your programs.
7. **Assemble applies to all files in directory.** Default value is **false**. If selected, the file currently open in the editor will become the "main" program in a multi-file assemble-and-link operation involving all assembly files (*.asm; *.s) in its directory. If successful, execution will begin with the currently open file.
8. **Assembler warnings are considered errors.** Default value is **false**. New in Release 3.5. If selected, the assemble operation will fail if any warnings are produced. At this time, all assembler warnings relate to unrecognized or ignored directives. MARS may be able to assemble code produced by compilers for other MIPS assemblers if this setting is deselected.
9. **Initialize Program Counter to global 'main' if defined.** Default value is **false**. New in Release 3.8. If selected, the Program Counter will be initialized to the address of the text segment statement with the global label 'main' if it exists. If it does not exist or if the setting is not selected, the Program Counter will be initialized to the default text segment starting address.
10. **Permit programs to use extended (pseudo) instructions and formats.** Default value is **true**. This includes all memory addressing modes other than the MIPS native mode (16 bit constant offset added to register content).
11. **Assemble and execute programs using delayed branching.** Default value is **false**. MIPS processors use delayed branches as part of the pipelined design, but it can be confusing to programmers. With delayed branching, the instruction following a branch or jump instruction *will always be executed* even if the branch condition is true! Assemblers and, failing that, programmers, often deal with this by following branches and jumps with a "`nop`" instruction. The MARS assembler does *not* insert a `nop`. When delayed branching was introduced in Release 3.3, the machine code generated for a branch instruction depended on this setting since its target value is relative to the

Program Counter (*PC-relative addressing*). Although technically correct, this led to confusion in the MARS community because the generated code did not match textbook examples. Starting with Release 3.4, the relative branching offset is always calculated as if delayed branching is enabled even when it is not. The runtime simulation adjusts accordingly.

12. **Self-modifying code.** Default value is **false**. New in Release 4.4. If selected, a running MIPS program can write to a user text segment address and can branch/jump to a user data segment address. These capabilities permit a program to dynamically generate and/or modify its binary code. Also permits interactive modification of text segment contents through either the Data Segment or Text Segment windows.
13. **The Editor dialog.** Use it to view and modify editor font settings. New with Release 3.3.
14. **The Highlighting dialog.** Use it to modify color and font settings for the highlighting of table items in the Text Segment window, Data Segment window, Registers window, Coprocessor0 window and Coprocessor1 window. Highlighting occurs during timed, stepped, and backstepped simulation. Color and font for normal (non-highlighted) display can also be set separately for even-numbered and odd-numbered display rows but not individually by windows. New with Release 3.6.
15. **The Exception Handler dialog.** It has the setting: Include this exception handler in all assemble operations. Default value is **false**. If selected, a button to browse to the desired file is enabled. New with Release 3.2
16. **The Memory Configuration dialog.** Use it to select from among available MIPS address space configurations. The default configuration is derived from SPIM; it was only one available from MARS 1.0 through MARS 3.6. New with Release 3.7.

Beginning with Release 3.2, settings are retained from one interactive session to the next. Settings are stored in a system-dependent way as specified by `java.util.prefs.Preferences`. Windows systems use the Registry. These settings are independent of command options given when using MARS from a command line; neither affects the other. We anticipate future releases will include additional settings and preferences.

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

SYSCALL functions available in MARS

Introduction

A number of system services, mainly for input and output, are available for use by your MIPS program. They are described in the table below.

MIPS register contents are not affected by a system call, except for result registers as specified in the table below.

How to use SYSCALL system services

Step 1. Load the service number in register $v0.
Step 2. Load argument values, if any, in $a0, $a1, $a2, or $f12 as specified.
Step 3. Issue the SYSCALL instruction.
Step 4. Retrieve return values, if any, from result registers as specified.

Example: display the value stored in $t0 on the console

```
li  $v0, 1           # service 1 is print integer
add $a0, $t0, $zero  # load desired value into argument register $a0, using pseudo-op
syscall
```

Table of Available Services

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	*See note below table*
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	*See note below table*
read character	12		$v0 contains character read

open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error). *See note below table*
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error). *See note below table*
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error). *See note below table*
close file	16	$a0 = file descriptor	
exit2 (terminate with value)	17	$a0 = termination result	*See note below table*
Services 1 through 17 are compatible with the SPIM simulator, other than Open File (13) as described in the Notes below the table. Services 30 and higher are exclusive to MARS.			
time (system time)	30		$a0 = low order 32 bits of system time $a1 = high order 32 bits of system time. *See note below table*
MIDI out	31	$a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127)	Generate tone and return immediately. *See note below table*
sleep	32	$a0 = the length of time to sleep in milliseconds.	Causes the MARS Java thread to sleep for (at least) the specified number of milliseconds. This timing will not be precise, as the Java implementation will add some overhead.
MIDI out synchronous	33	$a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127)	Generate tone and return upon tone completion. *See note below table*
print integer in hexadecimal	34	$a0 = integer to print	Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary.
print integer in binary	35	$a0 = integer to print	Displayed value is 32 bits, left-padding with zeroes if necessary.
print integer as unsigned	36	$a0 = integer to print	Displayed as unsigned decimal value.
(not used)	37-39		
set seed	40	$a0 = i.d. of pseudorandom number generator (any int). $a1 = seed for corresponding	No values are returned. Sets the seed of the corresponding underlying Java pseudorandom number generator `java.util.Random`

		pseudorandom number generator.	( ). *See note below table*
random int	41	$a0 = i.d. of pseudorandom number generator (any int).	$a0 contains the next pseudorandom, uniformly distributed int value from this random number generator's sequence. *See note below table*
random int range	42	$a0 = i.d. of pseudorandom number generator (any int). $a1 = upper bound of range of returned values.	$a0 contains pseudorandom, uniformly distributed int value in the range 0 <= [int] < [upper bound], drawn from this random number generator's sequence. *See note below table*
random float	43	$a0 = i.d. of pseudorandom number generator (any int).	$f0 contains the next pseudorandom, uniformly distributed float value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table*
random double	44	$a0 = i.d. of pseudorandom number generator (any int).	$f0 contains the next pseudorandom, uniformly distributed double value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table*
(not used)	45-49		
ConfirmDialog	50	$a0 = address of null-terminated string that is the message to user	$a0 contains value of user-chosen option 0: Yes 1: No 2: Cancel
InputDialogInt	51	$a0 = address of null-terminated string that is the message to user	$a0 contains int read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
InputDialogFloat	52	$a0 = address of null-terminated string that is the message to user	$f0 contains float read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
InputDialogDouble	53	$a0 = address of null-terminated string that is the message to user	$f0 contains double read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field
InputDialogString	54	$a0 = address of null-terminated string that is the message to user $a1 = address of input buffer $a2 = maximum number of characters to read	*See Service 8 note below table* $a1 contains status value 0: OK status. Buffer contains the input string. -2: Cancel was chosen. No change to buffer. -3: OK was chosen but no data had been input into field. No change to buffer. -4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null.
		$a0 = address of null-	

MessageDialog	55	terminated string that is the message to user $a1 = the type of message to be displayed: 0: error message, indicated by Error icon 1: information message, indicated by Information icon 2: warning message, indicated by Warning icon 3: question message, indicated by Question icon other: plain message (no icon displayed)	N/A
MessageDialogInt	56	$a0 = address of null-terminated string that is an information-type message to user $a1 = int value to display in string form after the first string	N/A
MessageDialogFloat	57	$a0 = address of null-terminated string that is an information-type message to user $f12 = float value to display in string form after the first string	N/A
MessageDialogDouble	58	$a0 = address of null-terminated string that is an information-type message to user $f12 = double value to display in string form after the first string	N/A
MessageDialogString	59	$a0 = address of null-terminated string that is an information-type message to user $a1 = address of null-terminated string to display after the first string	N/A

NOTES: Services numbered 30 and higher are not provided by SPIM
Service 8 - Follows semantics of UNIX 'fgets'. For specified length n, string can be no longer than n-1. If less than that, adds newline to end. In either case, then pads with null byte If n = 1, input is ignored and null byte placed at buffer address. If n < 1, input is ignored and nothing is written to the buffer.
Service 11 - Prints ASCII character corresponding to contents of low-order byte.
Service 13 - MARS implements three flag values: 0 for read-only, 1 for write-only with create, and 9 for write-only with create and append. It ignores mode. The returned file descriptor will be negative if the operation failed. The underlying file I/O implementation uses `java.io.FileInputStream.read()` to read and

`java.io.FileOutputStream.write()` to write. MARS maintains file descriptors internally and allocates them starting with 3. File descriptors 0, 1 and 2 are always open for: reading from standard input, writing to standard output, and writing to standard error, respectively (new in release 4.3).
Services 13,14,15 - In MARS 3.7, the result register was changed to $v0 for SPIM compatability. It was previously $a0 as erroneously printed in Appendix B of *Computer Organization and Design,*.
Service 17 - If the MIPS program is run under control of the MARS graphical interface (GUI), the exit code in $a0 is ignored.
Service 30 - System time comes from `java.util.Date.getTime()` as milliseconds since 1 January 1970.
Services 31,33 - Simulate MIDI output through sound card. Details below.
Services 40-44 use underlying Java pseudorandom number generators provided by the `java.util.Random` class. Each stream (identified by $a0 contents) is modeled by a different `Random` object. There are no default seed values, so use the Set Seed service (40) if replicated random sequences are desired.

Example of File I/O

The sample MIPS program below will open a new file for writing, write text to it from a memory buffer, then close it. The file will be created in the directory in which MARS was run.

```
# Sample MIPS program that writes to a new file.
#   by Kenneth Vollmar and Pete Sanderson

        .data
fout:   .asciiz "testout.txt"      # filename for output
buffer: .asciiz "The quick brown fox jumps over the lazy dog."
        .text
  ###############################################################
  # Open (for writing) a file that does not exist
  li   $v0, 13       # system call for open file
  la   $a0, fout     # output file name
  li   $a1, 1        # Open for writing (flags are 0: read, 1: write)
  li   $a2, 0        # mode is ignored
  syscall            # open a file (file descriptor returned in $v0)
  move $s6, $v0      # save the file descriptor
  ###############################################################
  # Write to file just opened
  li   $v0, 15       # system call for write to file
  move $a0, $s6      # file descriptor
  la   $a1, buffer   # address of buffer from which to write
  li   $a2, 44       # hardcoded buffer length
  syscall            # write to file
  ###############################################################
  # Close the file
  li   $v0, 16       # system call for close file
  move $a0, $s6      # file descriptor to close
  syscall            # close file
  ###############################################################
```

Using SYSCALL system services 31 and 33: MIDI output

These system services are unique to MARS, and provide a means of producing sound. MIDI output is simulated by your system sound card, and the simulation is provided by the `javax.sound.midi` package.

Service 31 will generate the tone then immediately return. Service 33 will generate the tone then sleep for the tone's duration before returning. Thus it essentially combines services 31 and 32.

This service requires four parameters as follows:

pitch ($a0)

- Accepts a positive byte value (0-127) that denotes a pitch as it would be represented in MIDI

Each number is one semitone / half-step in the chromatic scale.
- 0 represents a very low C and 127 represents a very high G (a standard 88 key piano begins at 9-A and ends at 108-C).
- If the parameter value is outside this range, it applies a default value 60 which is the same as middle C on a piano.
- From middle C, all other pitches in the octave are as follows:
  - 61 = C# or Db
  - 62 = D
  - 63 = D# or Eb
  - 64 = E or Fb
  - 65 = E# or F
  - 66 = F# or Gb
  - 67 = G
  - 68 = G# or Ab
  - 69 = A
  - 70 = A# or Bb
  - 71 = B or Cb
  - 72 = B# or C
- To produce these pitches in other octaves, add or subtract multiples of 12.

duration in milliseconds ($a1)

- Accepts a positive integer value that is the length of the tone in milliseconds.
- If the parameter value is negative, it applies a default value of one second (1000 milliseconds).

instrument ($a2)

- Accepts a positive byte value (0-127) that denotes the General MIDI "patch" used to play the tone.
- If the parameter is outside this range, it applies a default value 0 which is an *Acoustic Grand Piano*.
- General MIDI standardizes the number associated with each possible instrument (often referred to as *program change* numbers), however it does not determine how the tone will sound. This is determined by the synthesizer that is producing the sound. Thus a *Tuba* (patch 58) on one computer may sound different than that same patch on another computer.
- The 128 available patches are divided into instrument families of 8:

0-7	Piano	64-71	Reed
8-15	Chromatic Percussion	72-79	Pipe
16-23	Organ	80-87	Synth Lead
24-31	Guitar	88-95	Synth Pad
32-39	Bass	96-103	Synth Effects
40-47	Strings	104-111	Ethnic
48-55	Ensemble	112-119	Percussion
56-63	Brass	120-127	Sound Effects

- Note that outside of Java, General MIDI usually refers to patches 1-128. When referring to a list of General MIDI patches, 1 must be subtracted to play the correct patch. For a full list of General MIDI instruments, see www.midi.org/about-midi/gm/gm1sound.shtml. The General MIDI channel 10 percussion key map is not relevant to the toneGenerator method because it always defaults to MIDI channel 1.

volume ($a3)

- Accepts a positive byte value (0-127) where 127 is the loudest and 0 is silent. This value denotes MIDI velocity which refers to the initial attack of the tone.
- If the parameter value is outside this range, it applies a default value 100.
- MIDI velocity measures how hard a *note on* (or *note off*) message is played, perhaps on a

MIDI controller like a keyboard. Most MIDI synthesizers will translate this into volume on a logarithmic scale in which the difference in amplitude decreases as the velocity value increases.

- Note that velocity value on more sophisticated synthesizers can also affect the timbre of the tone (as most instruments sound different when they are played louder or softer).

System service 31 was developed and documented by Otterbein student Tony Brock in July 2007.

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Using MARS through its Integrated Development Environment (IDE)

The IDE is invoked when MARS is run with no command arguments, e.g. `java -jar mars.jar`. It may also be launched from a graphical interface by double-clicking the `mars.jar` icon that represents this executable JAR file. The IDE provides basic editing, assembling and execution capabilities. Hopefully it is intuitive to use. Here are comments on some features.

- **Menus and Toolbar**: Most menu items have equivalent toolbar icons. If the function of a toolbar icon is not obvious, just hover the mouse over it and a tool tip will soon appear. Nearly all menu items also have keyboard shortcuts. Any menu item not appropriate in a given situation is disabled.
- **Editor**: MARS includes two integrated text editors. The default editor, new in Release 4.0, features syntax-aware color highlighting of most MIPS language elements and popup instruction guides. The original, generic, text editor without these features is still available and can be selected in the Editor Settings dialog. It supports a single font which can be modified in the Editor Settings dialog. The bottom border of either editor includes the cursor line and column position and there is a checkbox to display line numbers. They are displayed outside the editing area. If you use an external editor, MARS provides a convenience setting that will automatically assemble a file as soon as it is opened. See the Settings menu.
- **Message Areas**: There are two tabbed message areas at the bottom of the screen. The *Run I/O* tab is used at runtime for displaying console output and entering console input as program execution progresses. You have the option of entering console input into a pop-up dialog then echoes to the message area. The *MARS Messages* tab is used for other messages such as assembly or runtime errors and informational messages. You can click on assembly error messages to select the corresponding line of code in the editor.
- **MIPS Registers**: MIPS registers are displayed at all times, even when you are editing and not running a program. While writing your program, this serves as a useful reference for register names and their conventional uses (hover mouse over the register name to see tool tips). There are three register tabs: the Register File (integer registers $0 through $31 plus LO, HI and the Program Counter), selected Coprocesor 0 registers (exceptions and interrupts), and Coprocessor 1 floating point registers.
- **Assembly**: Select *Assemble* from the *Run* menu or the corresponding toolbar icon to assemble the file currently in the Edit tab. Prior to Release 3.1, only one file could be assembled and run at a time. Releases 3.1 and later provide a primitive Project capability. To use it, go to the *Settings* menu and check *Assemble operation applies to all files in current directory.* Subsequently, the assembler will assemble the current file as the "main" program and also assemble all other assembly files (*.asm; *.s) in the same directory. The results are linked and if all these operations were successful the program can be executed. Labels that are declared global with the ".globl" directive may be referenced in any of the other files in the project. There is also a setting that permits automatic loading and assembly of a selected exception handler file. MARS uses the MIPS32 starting address for exception handlers: 0x80000180.
- **Execution**: Once a MIPS program successfully assembles, the registers are initialized and three windows in the Execute tab are filled: *Text Segment*, *Data Segment*, and *Program Labels*. The major execution-time features are described below.
- **Labels Window**: Display of the Labels window (symbol table) is controlled through the Settings menu. When displayed, you can click on any label or its associated address to center and highlight the contents of that address in the Text Segment window or Data Segment window as appropriate.

The assembler and simulator are invoked from the IDE when you select the *Assemble*, *Go*, or *Step* operations from the *Run* menu or their corresponding toolbar icons or keyboard shortcuts. MARS messages are displayed on the *MARS Messages* tab of the message area at the bottom of the screen. Runtime console input and output is handled in the *Run I/O* tab.

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

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Interactive Debugging Features

MARS provides many features for interactive debugging through its Execute pane. Features include:

- In *Step* mode, the next instruction to be simulated is highlighted and memory content displays are updated at each step.
- Select the *Go* option if you want to simulate continually. It can also be used to continue simulation from a paused (step, breakpoint, pause) state.
- Breakpoints are easily set and reset using the check boxes next to each instruction displayed in the Text Segment window. *New in Release 3.8:* You can temporarily suspend breakpoints using Toggle Breakpoints in the Run menu or by clicking the "Bkpt" column header in the Text Segment window. Repeat, to re-activate.
- When running in the *Go* mode, you can select the simulation speed using the Run Speed slider. Available speeds range from .05 instructions per second (20 seconds between steps) up to 30 instructions per second, then above this offers an "unlimited" speed. When using "unlimited" speed, code highlighting and memory display updating are turned off while simulating (but it executes really fast!). When a breakpoint is reached, highlighting and updating occur. Run speed can be adjusted while the program is running.
- When running in the *Go* mode, you can pause or stop simulation at any time using the *Pause* or *Stop* features. The former will pause execution and update the display, as if you were stepping or at a breakpoint. The latter will terminate execution and display final memory and register values. If running at "unlimited" speed, the system may not respond immediately but it will respond.
- You have the ability to interactively step "backward" through program execution one instruction at a time to "undo" execution steps. It will buffer up to 2000 of the most recent execution steps (this limit is stored in a properties file and can be changed). It will undo changes made to MIPS memory, registers or condition flags, but not console or file I/O. This should be a great debugging aid. It is available anytime execution is paused and at termination (even if terminated due to exception).
- When program execution is paused or terminated, select *Reset* to reset all memory cells and registers to their initial post-assembly values. In fact, Reset is implemented by re-assembling the program.
- Memory addresses and values, and register values, can be viewed in either decimal or hexadecimal format. All data are stored in little-endian byte order (each word consists of byte 3 followed by byte 2 then 1 then 0). Note that each word can hold 4 characters of a string and those 4 characters will appear in the reverse order from that of the string literal.
- Data segment contents are displayed 512 bytes at a time (with scrolling) starting with the data segment base address (0x10010000). Navigation buttons are provided to change the display to the next section of memory, the previous, or back to the initial (home) range. A combo box is also provided to view memory contents in the vicinity of the stack pointer (contents of MIPS $sp register), global pointer (contents of MIPS $gp register), the heap base address (0x10040000), .extern globals (0x10000000), the kernel data segment (0x90000000), or memory-mapped IO (MMIO, 0xFFFF0000). *Starting with Mars 4.4,* raw text segment contents can also be displayed.
- Contents of any data segment memory word and almost any MIPS register can be modified by editing its displayed table cell. Double-click on a cell to edit it and press the Enter key when finished typing the new value. If you enter an invalid 32-bit integer, the word INVALID appears in the cell and memory/register contents are not affected. Values can be entered in either decimal or hexadecimal (leading "0x"). Negative hexadecimal values can be entered in either two's complement or signed format. Note that three of the integer registers (zero, program

counter, return address) cannot be edited.
- *New in 4.4* If the setting for Self-Modifying Code is enabled (disabled by default, look in the Settings menu), text segment binary code can be modified using the same technique described above. It can also be modified by double-clicking on a cell in the Text Segment display's Code column.
- Contents of cells representing floating point registers can be edited as described above and will accept valid hexadecimal or decimal floating point values. Since each double-precision register overlays two single-precision registers, any changes to a double-precision register will affect one or both of the displayed contents of its corresponding single-precision registers. Changes to a single-precision register will affect the display of its corresponding double-precision register. Values entered in hexadecimal need to conform to IEEE-754 format. Values entered in decimal are entered using decimal points and E-notation (e.g. 12.5e3 is 12.5 times 10 cubed).
- Cell contents can be edited during program execution and once accepted will apply starting with the next instruction to be executed.
- Clicking on a Labels window entry will cause the location associated with that label to be centered and highlighted in the Text Segment or Data Segment window as appropriate. Note the Labels window is not displayed by default but can be by selecting it from the Settings menu.

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

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Using MARS from a command line.

MARS can be run from a command interpreter to assemble and execute a MIPS program in a batch fashion. The format for running MARS from a command line is:

```
java -jar mars.jar [options] program.asm [more files...] [ pa arg1 [more args...]]
```

Items in *[ ]* are optional. Valid options (not case sensitive, separated by spaces) are:

Option	Description	Since
`a`	assemble only, do not simulate	1.0
`ae`*n*	terminate MARS with integer exit code *n* if assembly error occurs	4.1
`ascii`	display memory or register contents interpreted as ASCII codes. (alternatives are `dec` and `hex`)	4.1
`b`	brief - do not display register/memory address along with contents	2.2
`d`	display MARS debugging statements (of interest mainly to MARS developer)	1.0
`db`	MIPS delayed branching is enabled.	3.3
`dec`	display memory or register contents in decimal. (alternatives are `ascii` and `hex`)	2.2
`dump`	dump memory contents to file. Option has 3 arguments, e.g. `dump <segment> <format> <file>`. Current supported segments are `.text` and `.data`. Also supports an address range (see *m-n* below). Current supported dump formats are `Binary`, `HexText`, `BinaryText`, `AsciiText`. See examples below.	3.4
`hex`	display memory or register contents in hexadecimal - this is the default. (alternatives are `ascii` and `dec`)	2.2
`h`	display this help. Use this option by itself and with no filename.	1.0
`ic`	display instruction count; the number of MIPS basic instructions 'executed'	4.3
`mc`	set memory configuration. Option has 1 argument, e.g. `mc <config>`. Argument `<config>` is case-sensitive and its possible values are `Default` for the default 32-bit address space, `CompactDataAtZero` for a 32KB address space with data segment at address 0, or `CompactTextAtZero` for a 32KB address space with text segment at address 0.	3.7
`me`	display MARS messages to standard err instead of standard out. Allows you to separate MARS messages from MIPS program output using redirection.	4.3
`nc`	copyright notice will not be displayed. Useful if redirecting or piping program output.	3.5
`np`	pseudo-instructions or extended instruction formats are not permitted.	3.0
	project option - will assemble the specified file and all other assembly files (*.asm; *.s) in its	

p	directory.	3.1
se*n*	terminate MARS with exit code *n* if simulate (run) error occurs	4.1
sm	start execution at statement having global label 'main' if defined	3.8
smc	Self Modifying Code - Program can write and execute in either text or data segment	4.4
we	assembler warnings will be considered errors.	3.5
n	where *n* is an integer maximum count of execution steps to simulate. If 0, negative or not specified, there is no maximum.	1.0
$*reg*	where *reg* is number or name (e.g. 5, t3, f10) of register whose content to display at end of run. Even-numbered float register displays both float and double. Option may be repeated. *NOTE: Depending on your command shell, you may need to escape the $, e.g.* `\$t3`	2.2
reg_name	where *reg_name* is the name (e.g. t3, f10) of register whose content to display at end of run. Even-numbered float register displays both float and double. Option may be repeated. $ not required.	2.2
m-n	memory address range from *m* to *n* whose contents to display at end of run. *m* and *n* may be decimal or hexadecimal (starts with `0x`), *m* <= *n*, both must be on word boundary. Option may be repeated.	2.2
pa	program arguments - all remaining space-separated items are argument values provided to the MIPS program via $a0 (argc - argument count) and $a1 (argv - address of array containing pointers to null-terminated argument strings). The count is also at the top of the runtime stack ($sp), followed by the array.*This option and its arguments must be the last items in the command!*	3.5

Example: `java -jar mars.jar h`
Displays command options and explanations.

Example: `java -jar mars.jar $s0 $s1 0x10010000-0x10010010 fibonacci.asm`
Assemble and run `fibonacci.asm`. At the end of the run, display the contents of registers `$s0` and `$s1`, and the contents of memory locations 0x10010000 through 0x10010010. The contents are displayed in hexadecimal format.

Example: `java -jar mars.jar a fibonacci.asm`
Assemble `fibonacci.asm`. Does not attempt to run the program, and the assembled code is not saved.

Example: `java -jar mars.jar 100000 infinite.asm`
Assemble and run `infinite.asm` for a maximum of 100,000 execution steps.

Example: `java -jar mars.jar p major.asm`
Assemble `major.asm` and all other files in the same directory, link the assembled code, and run starting with the first instruction in `major.asm`.

Example: `java -jar mars.jar major.asm minor.asm sub.asm`
Assemble and link `major.asm`, `minor.asm` and `sub.asm`. If successful, execution will begin with the first instruction in `major.asm`.

Example: `java -jar mars.jar a dump .text HexText hexcode.txt fibonacci.asm`
Assemble `fibonacci.asm` without simulating (note use of 'a' option). At end of assembly, dump the text segment (machine code) to file `hexcode.txt` in hexadecimal text format with one instruction per line.

Example: `java -jar mars.jar dump 0x10010000-0x10010020 HexText hexcode.txt fibonacci.asm`
Assemble and simulate `fibonacci.asm`. At end of simulation, dump the contents of addresses 0x1001000 to 0x10010020 to file `hexdata.txt` in hexadecimal text format with one word per line.

Example: `java -jar mars.jar t0 process.asm pa counter 10`
Assemble and run `process.asm` with two program argument values, "counter" and "10". It may retrieve the argument count (2) from `$a0`, and the address of an array containing pointers to the strings "count" and "10", from `$a1`. At the end of the run, display the contents of register `$t0`.

The ability to run MARS from the command line is useful if you want to develop scripts (macros) to exercise a given MIPS program under multiple scenarios or if you want to run a number of different MIPS programs such as for grading purposes.

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

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/`**.

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

MARS Release History

Mars 4.5 was released in August 2014. Enhancements and bug fixes include:

- The Keyboard and Display MMIO Simulator tool has been enhanced at the suggestion of Eric Wang at Washington State University. Until now, all characters written to the display via the Data Transmitter location (low order byte of memory word 0xFFFF000C) were simply streamed to the tools' display window. Mr. Wang requested the ability to treat the display window as a virtual text-based terminal by being able to programmatically clear the window or set the (x,y) position of a text cursor. Controlled placement of the text cursor (which is not displayed) allows you to, among other things, develop 2D text-mode games.
  - To clear the window, place ASCII/Unicode 12 decimal in the Data Transmitter byte. This is the non-printing Form Feed character.
  - To set the text cursor to a specified (x,y) position, where x is the column and y is the row, place ASCII/Unicode 7 in the Data Transmitter byte, and place the (x,y) position in the unused upper 24 bits of the Data Transmitter word. Place the X-position in bits 20-31 and the Y-position in bits 8-19. Position (0,0) is the upper-left corner of the display.
  - You can resize the display window to desired dimensions prior to running your MIPS program. Dimensions are dynamically displayed in the upper border. Note that the tool now contains a splitter between the display window and the keyboard window. Once the program is running, changes to the display size does not affect cursor positioning.

  The Help window for this tool is no longer modal, so you can view it while working in other windows. The Help window contains a lot of information so you will find it useful to be able to refer to it while working on your program.
- Installed the MIPS X-ray Tool developed by Marcio Roberto and colleagues at the Federal Center of Technological Education of Minas Gerais in Brazil. This tool animates a display of the MIPS datapath. The animation occurs while stepping through program execution. Search the Internet for "MIPS X-ray" to find relevant publications and other information.
- Context-sensitive help in the editor should now be easier to read. It was implemented as a menu of disabled items, which caused their text to be dimmed. The items are now enabled for greater visibility but clicking them will only make the list disappear.
- Bug Fix: Fixed an editor problem that affects certain European keyboards. The syntax-highlighting editor ignored the Alt key, which some European keyboards require to produce the # or $ characters in particular. I had no means of testing this, but Torsten Maehne in France send me a solution and Umberto Villano in Italy affirmed that it worked for him as well.
- Bug Fix: Source code references to Coprocessor 1 floating point registers (e.g. $f12) within macro definitions were erroneously flagged as syntax errors. MARS permits SPIM-style macro parameters (which start with $ instead of %) and did not correctly distinguish them from floating point register names. This has been fixed. Thanks to Rudolf Biczok in Germany for alerting me to the bug.
- Bug Fix: Corrected a bug that caused the Data Segment window to sometimes display incorrect values at the upper boundary of simulated memory segments. Thanks to Yi-Yu (James) Liu from Taiwan for alerting me to the bug, which was introduced in Mars 4.4.

Mars 4.4 was released in August 2013. Enhancements and bug fixes include:

- A feature to support self-modifying code has been developed by Carl Burch (Hendrix College) and Pete Sanderson. It is disabled by default and can be enabled through a Settings menu option. A program can write to the text segment and can also branch/jump to any user address in the data segments within the limits of the simulated address space. Text segment contents can also be edited interactively using the Data Segment window, and text segment contents within the address range of existing code can be edited interactively using the Text Segment window. In command mode, the smc option permits a program to write and execute in both text and data segments.
- Bug fix: An assembly error occurred when a line within a macro contained both a macro parameter and an identifier defined to have an .eqv substitution.
- Bug fix: If a macro name was used as a macro parameter, an assembly error occurred in some situations when a macro being used as an argument was defined following the macro that defined the parameter. The "for" macro described in the Macro help tab is an example.

Mars 4.3 was released in January 2013. Enhancements and bug fixes include:

- A macro facility has been developed by Mr. Mohammad Sekhavat. It is documented in the MIPS help tab Macros.
- A text substitution facility similar to #define has been developed using the new `.eqv` directive. It is also documented in the MIPS help tab Macros.
- A text insertion facility similar to #include has been developed using the new `.include` directive. It is also documented in the MIPS help tab Macros. It permits a macro to be defined in one file and included wherever needed.
- Two new command mode options are now available: ic (Instruction Count) to display a count of statements executed upon program termination, and me (Messages to Error) to send MARS messages to System.err instead of System.out. Allows you to separate MARS messages from MIPS output using redirection, if desired. Redirect a stream in DOS with "1>" or "2>" for out and err, respectively. To redirect both, use "> filename 2>&1"
- Changed the default font family settings from Courier New to Monospaced. This was in response to reports of Macs displaying left parentheses and vertical bars incorrectly.
- Changed the way operands for .byte and .half directives are range-checked. It will now work like SPIM, which accepts any operand value but truncates high-end bits as needed to fit the 1 (byte) or 2 (half) byte field. We'll still issue a warning but not an error.
- For file reads, syscall 14, file descriptor 0 is always open for standard input. For file writes, syscall 15, file descriptors 1 and 2 are always open for standard output and standard error, respectively. This permits you to write I/O code that will work either with files or standard I/O. When using the IDE, standard input and output are performed in the Run I/O tab. File descriptors for regular files are allocated starting with file descriptor 3.

Mars 4.2 was released in August 2011. Enhancements and bug fixes include:

- Performing a Save operation on a new file will now use the file's tab label as the the default filename in the Save As dialog (e.g. mips1.asm). Previously, it did not provide a default name.
- When the "assemble all files in directory" setting is enabled (useful for multi-file projects), you can now switch focus from one editor tab to another without invalidating the current assemble operation. You can similarly open additional project files. Previously, the open or tab selection would invalidate the assemble operation and any paused execution state or breakpoints would be lost.
- The Read String syscall (syscall 8) has been fortified to prevent Java exceptions from occurring when invalid values are placed in $a1.
- Will now perform runtime test for unaligned doubleword address in 'ldc1' and 'sdc1' instructions and trap if not aligned.
- Basic statements in the Text Segment display now renders immediates using eight hex digits when displaying in hex. Previously it rendered only four digits to conserve space. This led to confusing results. For instance, -1 and 0xFFFF would both be displayed as 0xFFFF but -1 expands to 0xFFFFFFFF and 0xFFFF expands to 0x0000FFFF.

Mars 4.1 was released in January 2011. Enhancements and bug fixes include:

- The ability to view Data Segment contents interpreted as ASCII characters has been added. You'll find it on the bottom border of the Data Segment Window as the checkbox "ASCII". This overrides the hexadecimal/decimal setting but only for the Data Segment display. It does not persist across sessions. Cells cannot be edited in ASCII format.
- The Dump Memory feature in the File menu now provides an ASCII dump format. Memory contents are interpreted as ASCII codes.
- A command-mode option "ascii" has been added to display memory or register contents interpreted as ASCII codes. It joins the existing "dec" and "hex" options for displaying in decimal or hexadecimal, respectively. Only one of the three may be specified.
- The actual characters to display for all the ASCII display options (data segment window, dump memory, command-mode option) are specified in the config.properties file. This includes a "placeholder" character to be displayed for all codes specified as non-printing. ASCII codes 1-7, 14-31, and 127-255 are specified as non-printing, but this can be changed in the properties file.
- A new Help tab called Exceptions has been added. It explains the basics of MIPS exceptions and interrupts as implemented in MARS. It also includes tips for writing and using exception handlers.
- A new Tool called Bitmap Display has been added. You can use it to simulate a simple bitmap display. Each word of the specified address space represents a 24 bit RGB color (red in bits 16-23, green in bits 8-15, blue in bits 0-7) and a word's value will be displayed on the Tool's display area when the word is written to by the MIPS program. The base address corresponds to the upper left corner of the display, and words are displayed in row-major order. Version 1.0 is pretty basic, constructed from the Memory Reference Visualization Tool code.
- Additional operand formats were added for the multiplication pseudo-instructions 'mul' and 'mulu'.
- The editor's context-sensitive pop-up help will now appear below the current line whenever possible. Originally it appeared either above, centered to the side, or below, depending on the current line's vertical position in the editing window. Displaying the pop-up above the current line tended to visually block important information, since frequently a line of code uses the same operand (especially registers) as the one immediately above it.
- The editor will now auto-indent each new line when the Enter key is pressed. Indentation of the new line will match that of the line that precedes it. This feature can be disabled in the Editor settings dialog.
- Two new command-mode options have been added. The "aeN" option, where N is an integer, will terminate MARS with exit value N when an assemble error occurs. The "seN" option will similarly terminate MARS with exit value N when a simulation (MIPS runtime) error occurs. These options can be useful when writing scripts for grading. Thanks to my Software Engineering students Robert Anderson, Jonathan Barnes, Sean Pomeroy, and Melissa Tress for designing and implementing these options.
- An editor bug that affected Macintosh users has been fixed. Command shortcuts, e.g. Command-s for save, did not function and also inserted the character into the text.
- A bug in Syscall 54 (InputDialogString) has been fixed. This syscall is the GUI equivalent of Syscall 8 (ReadString), which follows the semantics of UNIX 'fgets'. Syscall 54 has been modified to also follow the 'fgets' semantics.
- A bug in the Cache Simulator Tool has been fixed. The animator that paints visualized blocks green or red (to show cache hit or miss) sometimes paints the wrong block when set-associated caching is used. The underlying caching algorithm is correct so the numeric results such as hit ratios have always been correct. The animator has been corrected. Thanks to Andreas Schafer and his student Carsten Demel for bringing this to my attention.

Mars 4.0.1 was released in October 2010. It is a bug fix release to address three bugs.

- The Edit and Execute tabs of the IDE, which were relocated in 4.0 from the top to the left edge and oriented vertically, have been moved back to the top edge because Macintosh running Java 1.6 does not correctly render vertically-oriented tabs.
- An exception may be thrown in multi-file assembles when the last file of the assembly is not the longest. This occurs only when using the IDE, and has been corrected.
- If an assemble operation fails due to a non-existing exception handler file (specified through the IDE Settings menu), unchecking the exception handler setting does not prevent the same error from occuring on the next assemble. This has been corrected.

Mars 4.0 was released in August 2010. Enhancements and bug fixes include:

- *New Text Editor:* Mars features an entirely new integrated text editor. It creates a new tab for each file as it is opened. The editor now features language-aware color highlighting of many MIPS assembly language elements with customizable colors and styles. It also features automatic context-sensitive popup instruction guides. There are two levels: one with help and autocompletion of instruction names and a second with help information for operands. These and other new editor features can be customized or disabled through the expanded Editor Settings dialog. You can even revert to the previous notepad-style editor if you wish (multi-file capability is retained). The language-aware editor is based on the open source *jEdit Syntax Package* (syntax.jedit.org). It is separate from the assembler, so any syntax highlighting quirks will not affect assembly.
- *Improved Instruction Help:* All the instruction examples in the help tabs (and new popup instruction guides) now use realistic register names, e.g. $t1, $t2, instead of $1, $2. The instruction format key displayed above the MIPS help tabs has been expanded to include explanations of the various addressing modes for load and store instructions and pseudo-instructions. Descriptions have been added to every example instruction and pseudo-instruction.
- *Improved Assembly Error Capability:* If the assemble operation results in errors, the first error message in the Mars Messages text area will be highighted and the corresponding erroneous instruction will be selected in the text editor. In addition, you can click on any error message in the Mars Messages text area to select the corresponding erroneous instruction in the text editor. The first feature does not select in every situation (such as when assemble-on-open is set) but in the situations where it doesn't work no harm is done plus the second feature, clicking on error messages, can still be used.
- Console input syscalls (5, 6, 7, 8, 12) executed in the IDE now receive input keystrokes directly in the Run I/O text area instead of through a popup input dialog. Thanks to Ricardo Pascual for providing this feature! If you prefer the popup dialogs, there is a setting to restore them.
- The floor, ceil, trunc and round operations now all produce the MIPS default result 2^31-1 if the value is infinity, NaN or out of 32-bit range. For consistency, the sqrt operations now produce the result NaN if the operand is negative (instead of raising an exception). These cases are all consistent with FCSR (FPU Control and Status Register) Invalid Operation flag being off. The ideal solution would be to simulate the FCSR register itself so all MIPS specs for floating point instructions can be implemented, but that hasn't happened yet.
- The Basic column in the Text Segment Window now displays data and addresses in either decimal or hexadecimal, depending on the current settings. Note that the "address" in branch instructions is actually an offset relative to the PC, so is treated as data not address. Since data operands in basic instructions are no more than 16 bits long, their hexadecimal display includes only 4 digits.
- The Source column in the Text Segment Window now preserves tab spacing for a cleaner appearance (tab characters were previously not rendered).
- Instruction mnemonics can now be used as labels, e.g. "`b:`".
- New syscall 36 will display an integer as an unsigned decimal.
- A new tool, Digital Lab Sim, contributed by Didier Teifreto (dteifreto@lifc.univ-fcomte.fr). This tool features two seven-segment displays, a hexadecimal keypad, and a counter. It uses MMIO to explore interrupt-driven I/O in an engaging setting. More information is available from its Help feature. Many thanks!
- MARS 4.0 requires Java 1.5 (5.0) instead of 1.4. If this is an issue for you, let me know.

Mars 3.8 was released in January 2010. Enhancements and bug fixes include:

- A new feature to temporarily suspend breakpoints you have previously set. Use it when you feel confident enough to run your program without the breakpoints but not confident enough to clear them! Use the Toggle Breakpoints item in the Run menu, or simply click on the "Bkpt" column header in the Text Segment window. Repeat, to re-activate.
- Two new Tools contributed by Ingo Kofler of Klagenfurt University in Austria. One generates instruction statistics and the other simulates branch prediction using a Branch History Table.
- Two new print syscalls. Syscall 34 prints an integer in hexadecimal format. Syscall 35 prints an integer in binary format. Suggested by Bernardo Cunha of Portugal.
- A new Setting to control whether or not the MIPS program counter will be initialized to the statement with global label 'main' if such a statement exists. If the setting is unchecked or if checked and there is no 'main', the program counter will be initialized to the default starting address. Release 3.7 was programmed to automatically initialize

it to the statement labeled 'main'. This led to problems with programs that use the standard SPIM exception handler exceptions.s because it includes a short statement sequence at the default starting address to do some initialization then branch to 'main'. Under 3.7 the initialization sequence was being bypassed. By default this setting is unchecked. This option can be specified in command mode using the 'sm' (Start at Main) option.
- Mars Tools that exist outside of Mars can now be included in the Tools menu by placing them in a JAR and including it in a command that launches the Mars IDE. For example: `java -cp plugin.jar;Mars.jar Mars` Thanks to Ingo Kofler for thinking of this technique and providing the patch to implement it.
- Corrections and general improvements to the MIDI syscalls. Thanks to Max Hailperin of Gustavus Adolphus College for supplying them.
- Correction to an assembler bug that flagged misidentified invalid MIPS instructions as directives.

Mars 3.7 was released in August 2009. Enhancements and bug fixes include:

- A new feature for changing the address space configuration of the simulated MIPS machine. The 32-bit address space configuration used by all previous releases remains the default. We have defined two alternative configurations for a compact 32KB address space. One starts the text segment at address 0 and the other starts the data segment at address 0. A 32KB address space permits commonly-used load/store pseudo-instructions using labels, such as `lw $t0,increment`, to expand to a single basic instruction since the label's full address will fit into the 16-bit address offset field without sign-extending to a negative value. This was done in response to several requests over the years for smaller addresses and simplified expansions to make assembly programs easier to comprehend. This release does not include the ability to define your own customized configuration, although we anticipate adding it in the future. It is available both through the command mode (option mc) and the IDE. See `Memory Configuration...` at the bottom of the Settings menu.
- Related to the previous item: load and store pseudo-instructions of the form `lw $t0,label` and `lw $t0,label($t1)` will expand to a single instruction (`addi` for these examples) if the current memory configuration assures the label's full address will fit into the low-order 15 bits. Instructions for which this was implemented are: la, lw, lh, lb, lhu, lbu, lwl, lwr, ll, lwc1, ldc1, l.s, l.d, sw, sh, sb, swl, swr, sc, swc1, sdc1, s.s, and s.d.
- If a file contains a global statement label "main" (without quotes, case-sensitive), then execution will begin at that statement regardless of its address. Previously, program execution always started at the base address of the text segment. This will be handy for multi-file projects because you no longer need to have the "main file" opened in the editor in order to run the project. Note that main has to be declared global using the `.globl` directive.
- We have added a Find/Replace feature to the editor. This has been another frequent request. Access it through the Edit menu or Ctrl-F. Look for major enhancements to the editor in 2010!
- The syscalls for Open File (13), Read from File (14), and Write to File (15) all now place their return value into register $v0 instead of $a0. The table in *Computer Organization and Design*'s Appendix B on SPIM specifies $a0 but SPIM itself consistently uses $v0 for the return values.
- Pseudo-instructions for div, divu, mulo, mulou, rem, remu, seq, sne, sge, sgeu, sgt, sgtu, sle, sleu now accept a 16- or 32-bit immediate as their third operand. Previously the third operand had to be a register.
- Existing Tools were tested using reconfigured memory address space (see first item). Made some adaptations to the Keyboard and Display Simulator Tool that allow it to be used for Memory Mapped I/O (MMIO) even under the compact memory model, where the MMIO base address is 0x00007f00 instead of 0xffff0000. Highlighting is not perfect in this scenario.
- Bug Fix: The syscall for Open File (13) reversed the meanings of the terms *mode* and *flag*. Flags are used to indicate the intended use of the file (read/write). Mode is used to set file permissions in specific situations. MARS implements selected flags as supported by Java file streams, and ignores the mode if specified. For more details, see the Syscalls tab under Help. The file example in that tab has been corrected.
- Bug Fix: The assembler incorrectly generated an error on Jump instructions located in the kernel text segment.
- Bug Fix: The project (p) option in the command interface worked incorrectly when MARS was invoked within the directory containing the files to be assembled.
- Acknowledgement: The development of Release 3.7 was supported by a SIGCSE Special Projects Grant.

Mars 3.6 was released in January 2009. Enhancements and bug fixes include:

- We've finally implemented the most requested new feature: memory and register cells will be highlighted when written to during timed or stepped simulation! The highlighted memory/register cell thus represents the result of the instruction just completed. During timed or stepped execution, this is NOT the highlighted instruction. During back-stepping, this IS the highlighted instruction. The highlighted instruction is the next one to be executed in the normal (forward) execution sequence.
- In conjunction with cell highlighting, we've added the ability to customize the highlighting color scheme and font. Select Highlighting in the Settings menu. In the resulting dialog, you can select highlight background color, text color, and font for the different runtime tables (Text segment, Data segment, Registers). You can also select them for normal, not just highlighted, display by even- and odd-numbered row but not by table.
- Cool new Labels Window feature: the table can be sorted in either ascending or descending order based on either the Label (alphanumeric) or the Address (numeric) column. Just click on the column heading to select and toggle between ascending (upright triangle) or descending (inverted triangle). Addresses are sorted based on unsigned 32 bit values. The setting persists across sessions.
- The Messages panel, which includes the Mars Messages and Run I/O tabs, now displays using a mono-spaced (fixed character width) font. This facilitates text-based graphics when running from the IDE.
- The Mars.jar distribution file now contains all files needed to produce a new jar file. This will make it easier for you to expand the jar, modify source files, recompile and produce a new jar for local use. `CreatMarsJar.bat` contains the jar instruction.
- The Help window now includes a tab for Acknowledgements. This recognizes MARS contributors and correspondents.
- We've added a new system call (syscall) for generating MIDI tones synchronously, syscall 33. The original MIDI call returns immediately when the tone is generated. The new one will not return until the tone output is complete regardless of its duration.
- The Data Segment display now scrolls 8 rows (half a table) rather than 16 when the arrow buttons are clicked. This makes it easier to view a sequence of related cells that happen to cross a table boundary. Note you can hold down either button for rapid scrolling. The combo box with various data address boundaries also works better now.
- Bug Fix: Two corrections to the Keyboard and Display Simulator Tool. Transmitter Ready bit was not being reset based on instruction count when running in the kernel text segment, and the Status register's Exception Level bit was not tested before enabling the interrupt service routine (could lead to looping if interrupts occur w/i the interrupt service routine). Thanks to Michael Clancy and Carl Hauser for bringing these to my attention and suggesting solutions.
- Bug Fix: Stack segment byte addresses not on word boundaries were not being processed correctly. This applies to little-endian byte order (big-endian is not enabled or tested in MARS). Thanks to Saul Spatz for recognizing the problem and providing a patch.
- Minor Bug Fixes include: Correcting a fault leading to failure when launching MARS in command mode, clarifying assembler error message for too-few or too-many operands error, and correcting the description of lhu and lbu instructions from "unaligned" to "unsigned".

Mars 3.5 was released in August 2008. Major enhancements and bug fixes include:

- A new Tool, the Keyboard and Display MMIO Simulator, that supports polled and interrupt-driven input and output operations through Memory-Mapped I/O (MMIO) memory. The MIPS program writes to memory locations which serve as registers for simulated devices. Supports keyboard input and a simulated character-oriented display. Click the tool's Help button for more details.
- A new Tool, the Instruction Counter, contributed by MARS user Felipe Lessa. It will count the number of MIPS instructions executed along with percentages for R-format, I-format, and J-format instructions. Thanks, Felipe!
- Program arguments can now be provided to the MIPS program at runtime, through either an IDE setting or command mode. See the command mode "pa" option for more details on command mode operation. The argument count (argc) is placed in $a0 and the address of an array of null-terminated strings containing the arguments (argv) is placed in $a1. They are also available on the runtime stack ($sp).
- Two related changes permit MARS to assemble source code produced by certain compilers such as gcc. One change is to issue warnings rather than errors for unrecognized directives. MARS implements a limited number of directives. Ignore these warnings at your risk, but the assembly can continue. The second change is to allow

statement labels to contain, and specifically begin with, '$'.
- In command mode, final register values are displayed by giving the register name as an option. Register names begin with '$', which is intercepted by certain OS command shells. The convention for escaping it is not uniform across shells. We have enhanced the options so now you can give the register name without the '$'. For instance, you can use t0 instead of $t0 as the option. You cannot refer to registers by number in this manner, since an integer option is interpreted by the command parser as an instruction execution limit. Thanks to Lucien Chaubert for reporting this problem.
- Minor enhancements: The command mode dump feature has been extended to permit memory address ranges as well as segment names. If you enter a new file extension into the Open dialog, the extension will remain available throughout the interactive session. The data segment value repetition operator ':' now works for all numeric directives (.word, .half, .byte, .float, .double). This allows you to initialize multiple consecutive memory locations to the same value. For example:
```
ones: .half 1 : 8 # Store the value 1 in 8 consecutive halfwords
```
- Major change: Hexadecimal constants containing less than 8 digits will be interpreted as though the leading digits are 0's. For instance, 0xFFFF will be interpreted as 0x0000FFFF, not 0xFFFFFFFF as before. This was causing problems with immediate operands in the range 32768 through 65535, which were misinterpreted by the logical operations as signed 32 bit values rather than unsigned 16 bit values. Signed and unsigned 16 bit values are now distinguished by the tokenizer based on the prototype symbols -100 for signed and 100 for unsigned (mainly logical operations). Many thanks to Eric Shade of Missouri State University and Greg Gibeling of UC Berkeley for their extended efforts in helping me address this situation.
- Minor Bug Fixes: `round.w.s` and `round.w.d` have been modified to correctly perform IEEE rounding by default. Thanks to Eric Shade for pointing this out. Syscall 12 (read character) has been changed to leave the character in $v0 rather then $a0. The original was based on a misprint in Appendix A of *Computer Organization and Design*. MARS would not execute from the executable Mars.jar file if it was stored in a directory path those directory names contain any non-ASCII characters. This has been corrected. Thanks to Felipe Lessa for pointing this out and offering a solution. MARS will now correctly detect the EOF condition when reading from a file using syscall 14. Thanks to David Reimann for bringing this to our attention.

Mars 3.4.1 was released on 23 January 2008. It is a bug fix release to address two bugs.

- One bug shows up in pseudo-instructions in which the expansion includes branch instructions. The fixed branch offsets were no longer correct due to changes in the calculation of branch offsets in Release 3.4. At the same time, we addressed the issue of expanding such pseudo-instructions when delayed branching is enabled. Such expansions will now include a nop instruction following the branch.
- We also addressed an off-by-one error that occurred in generating the lui instruction in the expansion of conditional branch pseudo-instructions whose second operand is a 32 bit immediate.
- The expansions for a number of pseudo-instructions were modified to eliminate internal branches. These and other expansions were also optimized for sign-extended loading of 16-bit immediate operands by replacing the lui/ori or lui/sra sequence with addi. Pseudo-instructions affected by one or both of these modifications include: abs, bleu, bgtu, beq, bne, seq, sge, sgeu, sle, sleu, sne, li, sub and subi. These modifications were suggested by Eric Shade of Missouri State University.

Mars 3.4 was released in January 2008. Major enhancements are:

- A new syscall (32) to support pauses of specified length in milliseconds (sleep) during simulated execution.
- Five new syscalls (40-44) to support the use of pseudo-random number generators. An unlimited number of these generators are available, each identified by an integer value, and for each you have the ability to: set the seed value, generate a 32 bit integer value from the Java int range, generate a 32 bit integer value between 0 (inclusive) and a specified upper bound (exclusive), generate a 32-bit float value between 0 (inclusive) and 1 (exclusive), and generate a 64-bit double value between 0 (inclusive) and 1 (exclusive). All are based on the `java.util.Random` class.
- Ten new syscalls (50-59) to support message dialog windows and data input dialog windows. The latter are distinguished from the standard data input syscalls in that a prompting message is specified as a syscall argument and displayed in the input dialog. All are based on the `javax.swing.JOptionPane` class.

- The capability to dump `.text` or `.data` memory contents to file in various formats. The dump can be performed before or after program execution from either the IDE (File menu and toolbar) or from command mode. It can also be performed during an execution pause from the IDE. Look for the "Dump Memory" menu item in the File menu, or the "dump" option in command mode. A `.text` dump will include only locations containing an instruction. A `.data` dump will include a multiple of 4KB "pages" starting at the segment base address and ending with the last 4KB "page" to be referenced by the program. Current dump formats include pure binary (`java.io.PrintStream.write()` method), hexadecimal text with one word (32 bits) per line, and binary text with one word per line. An interface, abstract class, and format loader have been developed to facilitate development and deployment of additional dump formats. This capability was prototyped by Greg Gibeling of UC Berkeley.
- Changed the calculation of branch offsets when Delayed Branching setting is disabled. Branch instruction target addresses are represented by the relative number of words to branch. With Release 3.4, this value reflects delayed branching, regardless of whether the Delayed Branching setting is enabled or not. The generated binary code for branches will now match that of examples in the *Computer Organization and Design* textbook. This is a change from the past, and was made after extensive discussions with several MARS adopters. Previously, the branch offset was 1 lower if the Delayed Branching setting was enabled -- the instruction `label: beq $0,$0,label` would generate `0x1000FFFF` if Delayed Branching was enabled and `0x10000000` if it was disabled. Now it will generate `0x1000FFFF` in either case. The simulator will always branch to the correct location; MARS does not allow assembly under one setting and simulation under the other.
- Bug fix: The `mars.jar` executable JAR file can now be run from a different working directory. Fix was suggested by Zachary Kurmas of Grand Valley State University.
- Bug fix: The problem of MARS hanging while assembling a pseudo-instruction with a label operand that contains the substring "lab", has been fixed.
- Bug fix: No Swing-related code will be executed when MARS is run in command mode. This fixes a problem that occurred when MARS was run on a "headless" system (no monitor). Swing is the Java library to support programming Graphical User Interfaces. Fix was provided by Greg Gibeling of UC Berkeley.
- The '\0' character is now recognized when it appears in string literals.

MARS 3.3 was released in July 2007. Major enhancements are:

- Support for MIPS delayed branching. All MIPS computers implement this but it can be confusing for programmers, so it is disabled by default. Under delayed branching, the next instruction after a branch or jump instruction will *always* be executed, even if the branch or jump is taken! Many programmers and assemblers deal with this by inserting a do-nothing "nop" instruction after every branch or jump. The MARS assembler does *not* insert a "nop". Certain pseudo-instructions expand to a sequence that includes a branch; such instructions will not work correctly under delayed branching. Delayed branching is available in command mode with the "db" option.
- A new tool of interest mainly to instructors. The Screen Magnifier tool, when selected from the Tools menu, can be used to produce an enlarged static image of the pixels that lie beneath it. The image can be annotated by dragging the mouse over it to produce a scribble line. It enlarges up to 4 times original size.
- You now have the ability to set and modify the text editor font family, style and size. Select "Editor..." from the Settings menu to get the dialog. Click the Apply button to see the new settings while the dialog is still open. Font settings are retained from one session to the next. The font family list begins with 6 fonts commonly used across platforms (selected from lists found at www.codestyle.org), followed by a complete list. Two of the six are monospaced fonts, two are proportional serif, and two are proportional sans serif.
- The Labels window on the Execute pane, which displays symbol table information, has been enhanced. When you click on a label name or its address, the contents of that address are centered and highlighted in the Text Segment window or Data Segment window as appropriate. This makes it easier to set breakpoints based on text labels, or to find the value stored at a label's address.
- If you re-order the columns in the Text Segment window by dragging a column header, the new ordering will be remembered and applied from that time forward, even from one MARS session to the next. The Text Segment window is where source code, basic code, binary code, code addresses, and breakpoints are displayed.
- If a MIPS program terminates by "running off the bottom" of the program, MARS terminates, as before, without an exception, but now will display a more descriptive termination message in the Messages window. Previously, the termination message was the same as that generated after executing an Exit syscall.

- A new system call (syscall) to obtain the system time is now available. It is service 30 and is not available in SPIM. Its value is obtained from the `java.util.Date.getTime()` method. See the Syscall tab in MIPS help for further information.
- A new system call (syscall) to produce simulated MIDI sound through your sound card is now available. It is service 31 and is not available in SPIM. Its implementation is based on the `javax.sound.midi` package. It has been tested only under Windows. See the Syscall tab in MIPS help for further information.

MARS 3.2.1 was released in January 2007. It is a bug fix release that addresses the following bug in 3.2: a NullPointerException occurs when MIPS program execution terminates by "dropping off the bottom" of the program rather than by using one of the Exit system calls.

MARS 3.2 was released in December 2006. Major enhancements are:

- It fixes several minor bugs, including one that could cause incorrect file sequencing in the Project feature.
- It includes the `AbstractMarsToolAndApplication` abstract class to serve as a framework for easily constructing "tools" and equivalent free-standing applications that use the MARS assembler and simulator engines (kudos to the SIGCSE 2006 audience member who suggested this capability!). A subclass of this abstract class can be used both ways (tool or application).
- The floating point and data cache tools were elaborated in this release and a new tool for animating and visualizing memory references was developed. All are `AbstractMarsToolAndApplication` subclasses.
- This release includes support for exception handlers: the kernel data and text segments (.kdata and .ktext directives), the MIPS trap-related instructions, and the ability to automatically include a selected exception (trap) handler with each assemble operation.
- Items in the Settings menu became persistent with this release.
- Added default assembly file extensions "asm" and "s" to the Config.properties file and used those not only to filter files for the File Open dialog but also to filter them for the "assemble all" setting.
- Implemented a limit to the amount of text scrollable in the Mars Messages and Run I/O message tabs - default 1 MB is set in the Config.properties file.
- For programmer convenience, labels can now be referenced in the operand field of integer data directives (.word, .half, .byte). The assembler will substitute the label's address (low order half for .half, low order byte for .byte).
- For programmer convenience, character literals (e.g. 'b', '\n', '\377') can be used anywhere that integer literals are permitted. The assembler converts them directly to their equivalent 8 bit integer value. Unicode is not supported and octal values must be exactly 3 digits ranging from '\000' to '\377'.
- Replaced buttons for selecting Data Segment display base addresses with a combo box and added more base addresses: MMIO (0xFFFF0000), .kdata (0x90000000), .extern (0x10000000).

MARS 3.1 was released in October 2006. The major issues and features are listed here:

- It addressed several minor limits (Tools menu items could not be run from the JAR file, non-standard shortcuts for Mac users, inflexible and sometimes inappropriate sizing of GUI components).
- It changed the way SYSCALLs are implemented, to allow anyone to define new customized syscall services without modifying MARS.
- It added a primitive Project capability through the "Assemble operation applies to all files in current directory." setting (also available as "p" option in command mode). The command mode also allows you to list several file names not necessarily in the same directory to be assembled and linked.
- Multi-file assembly also required implementing the ".globl" and ".extern" directives.
- And although "Mars tools" are not officially part of MARS releases, MARS 3.1 includes the initial versions of two tools: one for learning about floating point representation and another for simulating data caching capability.

MARS 3.0 was released in August 2006, with one bug fix and two major additions.

- The bug fix was corrected instruction format for the slti and sltiu instructions.
- One major addition is a greatly expanded MIPS-32 instruction set (trap instructions are the only significant ones to remain unimplemented). This includes, via pseudo-instructions, all reasonable memory addressing modes for

the load/store instructions.
- The second major addition is ability to interactively step "backward" through program execution one instruction at a time to "undo" execution steps. It will buffer up to 2000 of the most recent execution steps (this limit is stored in a properties file and can be changed). It will undo changes made to MIPS memory, registers or condition flags, but not console or file I/O. This should be a great debugging aid. It is available anytime execution is paused and at termination (even if terminated due to exception).
- A number of IDE settings, described above, are now available through the Settings menu.

MARS 2.2 was released in March 2006 with additional bug fixes and implemented command line options (run MARS from command line with h option for command line help). This also coincides with our SIGCSE 2006 paper "MARS: An Education-Oriented MIPS Assembly Language Simulator".

MARS 2.1 was released in October 2005 with some bug fixes.

MARS 2.0 was released in September 2005. It incorporated significant modifications to both the GUI and the assembler, floating point registers and instructions most notably.

MARS 1.0 was released in January 2005 and publicized during a poster presentation at SIGCSE 2005.

Dr. Ken Vollmar initiated MARS development in 2002 at Missouri State University. In 2003, Dr. Pete Sanderson of Otterbein College and his student Jason Bumgarner continued implementation. Sanderson implemented the assembler and simulator that summer, and the basic GUI the following summer, 2004.

The development of Releases 3.1 and 3.2 in 2006 and 4.0 in 2010 were supported by the Otterbein College sabbatical leave program. The development of Release 3.7 during summer 2009 was supported by a SIGCSE Special Projects Grant.

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

Intro ***Settings*** ***Syscalls*** ***IDE*** ***Debugging*** ***Command*** ***Tools*** ***History*** ***Limitations***
Exception Handlers ***Macros*** ***Acknowledgements*** ***MARS home***

MARS - Mips Assembly and Runtime Simulator

Release 4.5

August 2014

Operating Requirements

MARS is written in Java and requires at least Release 1.5 of the J2SE Java Runtime Environment (JRE) to work. The graphical IDE is implemented using Swing. It has been tested on Windows XP, Vista and 7; Mac OS X; and is also being used under Linux.

Some MARS Assembler and Simulator Limitations

Releases 3.0 and later assemble and simulate nearly all the MIPS32 instructions documented in the textbook *Computer Organization and Design, Fourth Edition* by Patterson and Hennessy, Elsevier - Morgan Kaufmann, 2009. All basic and pseudo instructions, directives, and system services described in Appendix B are implemented.

Limitations of MARS as of Release 4.4 include:

- Memory segments (text, data, stack, kernel text, kernel data) are limited to 4MB each starting at their respective base addresses.
- There is no pipelined mode (but delayed branching is supported).
- If you open a file which is a link or shortcut to another file, MARS will *not* open the target file. The file open dialog is implemented using Java Swing's JFileChooser, which does not support links.
- Very few configuration changes, besides those in the Settings menu, are saved from one session to the next. The editor settings, which include font settings and display of line numbers, are saved.
- The IDE will work only with the MARS assembler. It cannot be used with any other compiler, assembler, or simulator. The MARS assembler and simulator can be used either through the IDE or from a command prompt.
- *Bug:*The error message highlighter does not automatically select the code for the first assembly error if the file containing the error is not open at the time of assembly (assemble-on-open, assemble-all).
- *Bug:* The Screen Magnifier screen capture feature does not appear to work properly under Windows Vista.
- *Bug:* There appears to be a memory leak in the Editor. Several different people have independently reported the same behavior: severe slowdown in editor response during an extended interactive session. If MARS is exited and restarted, this behavior disappears and the editor responds instantly to actions.
- *Not a bug, but documented here anyway:* MIPS Branch instruction target addresses are represented by the relative number of words to branch. With Release 3.4, this value reflects delayed branching, regardless of whether the Delayed Branching setting is enabled or not. The generated binary code for branches will now match that of examples in the *Computer Organization and Design* textbook. This is a change from the past, and was made after extensive discussions with several faculty adopters of MARS. Previously, the branch offset was 1 lower if the Delayed Branching setting was enabled -- the instruction `label: beq $0,$0,label` would generate `0x1000FFFF` if Delayed Branching was enabled and `0x10000000` if it was disabled. Now it will generate `0x1000FFFF` in either case. The simulator will always branch to the correct location; MARS does not allow assembly under one setting and simulation under the other to occur.

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

Writing and Using MIPS exception handlers in MARS

Introduction

Exception handlers, also known as *trap handlers* or *interrupt handlers*, can easily be incorporated into a MIPS program. This guide is not intended to be comprehensive but provides the essential information for writing and using exception handlers.

Although the same mechanism services all three, *exceptions*, *traps* and *interrupts* are all distinct from each other. Exceptions are caused by exceptional conditions that occur at runtime such as invalid memory address references. Traps are caused by instructions constructed especially for this purpose, listed below. Interrupts are caused by external devices.

MARS partially but not completely implements the exception and interrupt mechanism of SPIM.

Essential Facts

Some essential facts about writing and using exception handlers include:

- MARS simulates basic elements of the MIPS32 exception mechanism.
- The MIPS instruction set includes a number of instructions that conditionally trigger a trap exception based on the relative values of two registers or of a constant and a register: `teq`, `teqi` (trap if equal), `tne`, `tnei` (trap if not equal), `tge`, `tgeu`, `tgei`, `tgeiu` (trap if greater than or equal), `tlt`, `tltu`, `tlti`, `tltiu` (trap if less than)
- When an exception occurs,
  1. Coprocessor 0 register $12 (status) bit 1 is set
  2. Coprocessor 0 register $13 (cause) bits 2-6 are set to the exception type (codes below)
  3. Coprocessor 0 register $14 (epc) is set to the address of the instruction that triggered the exception
  4. If the exception was caused by an invalid memory address, Coprocessor 0 register $8 (vaddr) is set to the invalid address.
  5. Execution flow jumps to the MIPS instruction at memory location `0x800000180`. This address in the kernel text segment (`.ktext` directive) is the standard MIPS32 exception handler location. The only way to change it in MARS is to change the MIPS memory configuration through the Settings menu item Memory Configuration.
- There are three ways to include an exception handler in a MIPS program
  1. Write the exception handler in the same file as the regular program. An example of this is presented below.
  2. Write the exception handler in a separate file, store that file in the same directory as the regular program, and select the Settings menu item "Assemble all files in directory"
  3. Write the exception handler in a separate file, store that file in any directory, then open the "Exception Handler..." dialog in the Settings menu, check the check box and browse to that file.
- If there is no instruction at location `0x800000180`, MARS will terminate the MIPS program with an appropriate error message.
- The exception handler can return control to the program using the `eret` instruction. This will place the EPC register $14 value into the Program Counter, so be sure to increment $14 by 4 before returning to skip over the instruction that caused the exception. The `mfc0` and `mtc0` instructions are used to read from and write to Coprocessor 0 registers.
- Bits 8-15 of the Cause register $13 can also be used to indicate pending interrupts. Currently this is used only by the Keyboard and Display Simulator Tool, where bit 8 represents a keyboard interrupt and bit 9 represents a display interrupt. For more details, see the Help panel for that Tool.
- Exception types declared in `mars.simulator.Exceptions`, but not necessarily implemented, are

ADDRESS_EXCEPTION_LOAD (4), ADDRESS_EXCEPTION_STORE (5), SYSCALL_EXCEPTION (8), BREAKPOINT_EXCEPTION (9), RESERVED_INSTRUCTION_EXCEPTION (10), ARITHMETIC_OVERFLOW_EXCEPTION (12), TRAP_EXCEPTION(13), DIVIDE_BY_ZERO_EXCEPTION (15), FLOATING_POINT_OVERFLOW (16), and FLOATING_POINT_UNDERFLOW (17).

- When writing a non-trivial exception handler, your handler must first save general purpose register contents, then restore them before returning.

Example of Trap Handler

The sample MIPS program below will immediately generate a trap exception because the trap condition evaluates true, control jumps to the exception handler, the exception handler returns control to the instruction following the one that triggered the exception, then the program terminates normally.

```
   .text
main:
   teqi $t0,0     # immediately trap because $t0 contains 0
   li   $v0, 10   # After return from exception handler, specify exit service
   syscall        # terminate normally

# Trap handler in the standard MIPS32 kernel text segment

   .ktext 0x80000180
   move $k0,$v0   # Save $v0 value
   move $k1,$a0   # Save $a0 value
   la   $a0, msg  # address of string to print
   li   $v0, 4    # Print String service
   syscall
   move $v0,$k0   # Restore $v0
   move $a0,$k1   # Restore $a0
   mfc0 $k0,$14   # Coprocessor 0 register $14 has address of trapping instruction
   addi $k0,$k0,4 # Add 4 to point to next instruction
   mtc0 $k0,$14   # Store new address back into $14
   eret           # Error return; set PC to value in $14
   .kdata
msg:
   .asciiz "Trap generated"
```

Widely Used Exception Handler

The exception handler `exceptions.s` provided with the SPIM simulator will assemble and run under MARS. The MARS assembler will generate warnings because this program contains directives that it does not recognize, but as long as the setting "Assembler warnings are considered errors" is *not* set this will not cause any problems.

Writing and Using Macros

`.macro`, `.end_macro`,`.eqv` and `.include` directives are new in MARS 4.3

Introduction to macros

Patterson and Hennessy define a **macro** as *a pattern-matching and replacement facility that provides a simple mechanism to name a frequently used sequence of instructions* [1]. This permits the programmer to specify the instruction sequence by invoking the macro. This requires only one line of code for each use instead of repeatedly typing in the instruction sequence each time. It follows the axiom "define once, use many times," which not only reduces the chance for error but also facilitates program maintenance.

Macros are like procedures (subroutines) in this sense but operate differently than procedures. Procedures in MIPS assembly language follow particular protocols for procedure definition, call and return. Macros operate by substituting the macro body for each use at the time of assembly. This substitution is called *macro expansion.*. They do not require the protocols and execution overhead of procedures.

As a simple example, you may want to terminate your program from a number of locations. If you are running from the MARS IDE, you will use system call 10, `exit`. The instruction sequence is pretty easy

```
li $v0,10
syscall
```

but still tedious. You can define a macro, let's call it `done`, to represent this sequence

```
.macro done
li $v0,10
syscall
.end_macro
```

then invoke it whenever you wish with the statement

```
done
```

At assembly time, the assembler will replace each occurrence of the statement `done` with the two-statement sequence

```
li $v0,10
syscall
```

This is the macro expansion. The runtime simulator is unaware of macros or macro expansion.

If running MARS from the command line, perhaps you want to return a termination value. This can be done with syscall 17, `exit2`, which takes the termination value as an argument. An equivalent macro, let's call it `terminate` would be

```
.macro terminate (%termination_value)
li $a0, %termination_value
li $v0, 17
syscall
.end_macro
```

This macro defines a *formal parameter* to represent the termination value. You would invoke it with the statement

```
terminate (1)
```

to terminate with value 1. Upon assembly, the statement `terminate (1)` would be replaced by the three-statement sequence

```
li $a0, 1
```

```
                li $v0, 17
                syscall
```

The *argument value*, 1, is substituted wherever the formal parameter `%termination_value` appears in the macro body. This is a textual substitution. Note that in this example the argument value must be an integer, not a register name or a label, because the parameter is used as the second operand in the Load Immediate operation.

In MARS, a macro is similar to an extended (pseudo) instruction. They are distinguished in that the expansion of extended instructions is supported by an internally-defined specification language and mechanism which can manipulate argument values. The macro facility can only substitute argument values as given, and it uses a separate mechanism from extended instructions.

Additional examples and details follow.

How to define macros

The first line begins with a `.macro` directive followed by an optional list of formal parameters. Placing commas between parameters and parentheses around the list is optional.

Each formal parameter is an identifier that begins with a `%` character. For compatibility with the SPIM preprocessor APP, it may alternatively begin with `$`.

The lines that follow define the body of the macro. Use the formal parameters as appropriate. The body may contain data segments as well as text segments.

The macro definition finishes with a `.end_macro` directive.

See the Notes below for additional information.

How to use macros

To invoke a macro, form a statement consisting of the macro name and then one token for each argument to be substituted for its corresponding formal parameter by the assembler. The argument list may optionally be surrounded by parentheses. Arguments may be separated either by spaces or commas.

Macro expansion is a pre-processing task for assemblers.

Notes

- A macro definition must appear before its use. No forward references.
- All macro definitions are local in each file and they cannot be global.
- Nested macro definitions are not supported. No `.macro` directive should appear inside body of a macro definition.
- A macro definition can contain a call to a previously-defined macro. Only backward references are allowed.
- Labels placed in the body of a macro definition will not have same name after macro expansion. During expansion, their name will be followed by "_M#" where # will be a unique number for each macro expansion.
- Two macros with the same name but different number of parameters are considered different and both can be used.
- A macro defined with the same name and same number of parameters as another macro defined before it will be ignored.
- Each argument in a macro call can only be a single language element (token). For instance "4($t0)" cannot be an argument.
- Macros are a part of the assembler, not the ISA. So the syntax might be different with other assemblers. For compatibility with the SPIM simulator, *SPIM-style macros are also supported in MARS*. SPIM-style macros are same as MARS but formal parameters begin with "$" instead of "%".

Examples

- Printing an integer (argument may be either an immediate value or register name):

```
        .macro print_int (%x)
        li $v0, 1
        add $a0, $zero, %x
        syscall
        .end_macro

        print_int ($s0)
        print_int (10)
```

- Printing a string (macro will first assign a label to its parameter in data segment then print it):

```
        .macro print_str (%str)
        .data
myLabel: .asciiz %str
        .text
        li $v0, 4
        la $a0, myLabel
        syscall
        .end_macro

        print_str ("test1")      #"test1" will be labeled with name "myLabel_M0"
        print_str ("test2")      #"test2" will be labeled with name "myLabel_M1"
```

- Implementing a simple for-loop:

```
        # generic looping mechanism
        .macro for (%regIterator, %from, %to, %bodyMacroName)
        add %regIterator, $zero, %from
        Loop:
        %bodyMacroName ()
        add %regIterator, %regIterator, 1
        ble %regIterator, %to, Loop
        .end_macro

        #print an integer
        .macro body()
        print_int $t0
        print_str "\n"
        .end_macro

        #printing 1 to 10:
        for ($t0, 1, 10, body)
```

  The `for` macro has 4 parameters. `%regIterator` should be the name of a register which iterates from `%from` to `%to` and in each iteration `%bodyMacroName` will be expanded and run. Arguments for `%from` and `%to` can be either a register name or an immediate value, and `%bodyMacroName` should be name of a macro that has no parameters.

Macro source line numbers

For purpose of error messaging and Text Segment display, MARS attempts to display line numbers for both the definition and use of the pertinent macro statement. If an error message shows the line number in the form "`X->Y`" (e.g. "`20->4`"), then `X` is the line number in the expansion (use) where the error was detected and `Y` is the line number in the macro definition. In the Text Segment display of source code, the macro definition line number will be displayed within brackets, e.g. "`<4>`", at the point of expansion. Line numbers should correspond to the numbers you would see in the text editor.

The .eqv directive

The `.eqv` directive (short for "equivalence") is also new in MARS 4.3. It is similar to `#define` in C or C++. It is used to substitute an arbitrary string for an identifier. It is useful but much less powerful than macros. It was developed independently of the macro facility.

```
    .eqv
```

Using , you can specify simple substitutions that provide "define once, use many times" capability at assembly pre-processing time. For example, once you define

```
        .eqv  LIMIT      20
        .eqv  CTR        $t2
        .eqv  CLEAR_CTR  add  CTR, $zero, 0
```

then you can refer to them in subsequent code:

```
        li $v0,1
        CLEAR_CTR
loop:   move $a0, CTR
        syscall
        add   CTR, CTR, 1
        blt   CTR, LIMIT, loop
        CLEAR_CTR
```

During assembly pre-processing, the `.eqv` substitutions will be applied. The resulting code is

```
        li   $v0,1
        add  $t2, $zero, 0
loop:   move $a0, $t2
        syscall
        add  $t2, $t2, 1
        blt  $t2, 20, loop
        add  $t2, $zero, 0
```

which when run will display the values 0 through 19 on one line with no intervening spaces.

Note that the substitution string is not limited to a single token. Like `.macro`, `.eqv` is local to the file in which it is defined, and must be defined prior to use. Macro bodies can contain references to `.eqv` directives.

The .include directive

The `.include` directive is also new in MARS 4.3. It has one operand, a quoted filename. When the directive is carried out, the contents of the specified file are substituted for the directive. This occurs during assembly preprocessing. It is like `#include` in C or C++.

`.include` is designed to make macro and equivalence (.eqv directive) use more convenient. Both macro definitions and equivalence definitions are *local*, which means they can be used only in the same file where defined. Without `.include`, you would have to repeat their definitions in every file where you want to use them. Besides being tedious, this is poor programming practice; remember "define once, use many times." Now you can define macros and equivalences in a separate file, then include it in any file where you want to use them.

The `.include` preprocessor will detect and flag any circular includes (file that includes itself, directly or indirectly).

The use of `.include` presents some challenges for error messaging and for source code numbering in the Text Segment display. If a file being included has any assembly errors, the filename and line number in the error message should refer to the file being included, not the file it was substituted into. Similarly, the line number given in the Text Segment source code display refers to the line in the file being included. Thus the displayed line numbers do not monotonically increase - this is also the case when using the "assemble all" setting. Line numbers should correspond to the numbers you would see in the text editor.

As a simple example, you could define the `done` macro (and others) in a separate file then include it wherever you need it. Suppose "macros.asm" contains the following:

```
                  .macro done
                  li $v0,10
                  syscall
                  .end_macro
```

You could then include it in a different source file something like this:

```
                .include "macros.asm"
                .data
value:          .word    13
                .text
                li       $v0, 1
                lw       $a0, value
                syscall
                done
```

During assembly preprocessing, this would be expanded to

```
                .macro done
                li $v0,10
                syscall
                .end_macro
                .data
value:          .word    13
                .text
                li       $v0, 1
                lw       $a0, value
                syscall
                done
```

The assembler will then perform the appropriate macro expansion.

Acknowledgements

The MARS macro facility was developed in 2012 by Mohammad Hossein Sekhavat, sekhavat17@gmail.com, while an engineering student at Sharif University in Tehran. MARS creators Pete and Ken are incredibly grateful for his contribution! Pete developed `.eqv` and `.include` at about the same time.

References

[1] *Computer Organization and Design: The Hardware/Software Interface, Fourth Edition,* Patterson and Hennessy, Morgan Kauffman Publishers, 2009.

MARS Acknowledgements

Updated 18 August 2014

Pete and Ken would like to acknowledge the many helpful contributors to MARS. It has succeeded beyond our wildest expectations and for this we are most grateful. Its success would not be possible without your feedback, suggestions and assistance!

We are pleased to recognize these contributions to release 4.5:

- **Torsten Mahne**, **Umberto Villano** and others who took care of the bug with certain European keyboards that require an Alt key combo to form essential MIPS assembly characters like $ and #. I had no means of testing it.
- **Eric Wang** at Washington State University, who suggested adding cursor positioning to the Keyboard and Display MMIO Simulator tool.
- **Marcio Roberto** and everyone else involved in the development of MIPS X-Ray tool, which has been around for several years but only now added to the release. Sorry for the delay.

We also appreciate the contributions others have made to previous releases:

- **Carl Burch** of Hendrix College, who developed the mechanism for simulating the execution of straight binary code. Previously, execution was based on ProgramStatement objects generated by the assembler. This, combined with the added capabilities to write to the text segment and branch/jump into the data segment at runtime, permits one to produce self-modifying programs, simulate buffer overflow attacks, and the like.
- **Tom Bradford**, **Slava Pestov** and others, who developed the jEdit Syntax Package (syntax.jedit.org) at the heart of the syntax-aware color highlighting editor. It was old but the licensing was right and it was written for embedding into Java applications.
- **Mohammad Sekhavat** from Sharif University in Tehran, who developed the macro capability.
- **Greg Gibeling** of UC Berkeley, who introduced capabilities into his customized version of MARS that have subsequently been expanded and integrated into our release. These include the ability to dump MIPS memory contents to file and parser improvements to distinguish signed from unsigned hexadecimal constants.
- **Eric Shade** of Missouri State University, who suggested several improvements to pseudo-instruction expansions such as elimination of internal branches and improvements to the sign-extended loading of 16-bit immediate operands.
- **Saul Spatz** of the University of Missouri Kansas City, who noticed and provided a solution for a flaw in the calculation of byte-oriented addresses in the simulated MIPS memory stack segment. He has also suggested several improvements that we have implemented.
- **Zachary Kurmas** of Grand Valley State University, who suggested several bug fixes and who encorporated MARS into his own successful `JLSCircuitTester` digital logic simulator software.
- **Felipe Lessa**, who contributed the Instruction Counter tool and suggested a solution for the problem of MARS inability to launch when stored in a directory whose name contained non-ASCII characters.
- **Carl Hauser** of Washington State University, who pointed out and provided a solution to a flaw in the Keyboard and Display Simulator Tool in how it used the Exception Level bit in the Coprocessor1 Status register. Also thanks to **Michael Clancy** of UC Berkeley for pointing out a flaw in the tool's mechanism for resetting the Transmit Ready bit when operating in kernel memory.
- **Dave Poplawski** of Michigan Technological University, for his assistance in working through some issues with signed/unsigned constants and with output redirection.
- **Ingo Kofler** of Klagenfurt University in Austria, who contributed two Tools: a tool to collect Instruction Statistics, and a tool to simulate branch prediction with a Branch History Table.
- **Brad Richards** and **Steven Canfield** from the University of Puget Sound, for providing a technique that improved file loading performance.

- **Jan Schejbal** and **Jan-Peter Kleinhans** of Darmstadt technical university in Germany, for suggesting and providing a patch to display Run I/O text in a constant-width font.
- **Max Hailperin** of Gustavus Adolphus College, who made several improvements to the MIDI syscalls.
- **David Patterson** of UC Berkeley, for making time in his busy schedule for Ken's demo of MARS.
- **Denise Penrose** and **Nate McFadden** of Morgan Kaufmann Publishers, for their assistance as editors of *Computer Organization and Design*.
- **Ricardo Pascual** of University of Murcia in Spain, who contributed the code to permit input syscall keystrokes to be typed directly into the Run I/O window instead of through a popup dialog.
- **Didier Teifreto** of UniversitÃ© de Franche-ComtÃ© in France, who contributed the Digital Lab Sim tool.
- **Facundo Agriel** of the University of Illinois at Chicago, who added font selection to the Keyboard and Display Simulator tool.
- Patrik Lundin for contributing the code to add scrolling to the keyboard and display simulator.
- Otterbein students Robert Anderson, Jonathan Barnes, Sean Pomeroy and Melissa Tress for contributing the new command-mode options for specifying MARS exit codes when assembly or simulation errors occur. This was sparked by a comment from Zheming Jim of the University of South Carolina.
- The unknown audience member at our SIGCSE 2006 conference presentation, who suggested that MARS would also be useful running in the background in support of an external application. This led directly to our development of the Tools framework and API that truly distinguishes MARS from all other MIPS simulators.

We would also like to recognize many others who have contacted us to point out bugs, suggest improvements, or engaged us in interesting correspondence. The bugs have been addressed and the improvements either implemented or added to our wish list. Correspondents include: William Bai, Miguel Barao, James Baltimore, Jared Barneck, Bruce Barton, Rudolf Biczok, Battista Biggio, Carl Burch, Ram Busani, Gene Chase, Lucien Chaubert, David Chilson, Sangyeun Cho, Donald Cooley, Bernardo Cunha, John Donaldson, Abhik Ghosh, Michael Grant, Thomas Hain, John Ham, Kurtis Hardy, Justin Harlow, David Harris, Bill Hsu, Pierre von Kaenel, Amos Kittelson, klondike, Geoffrey Knauth, Sudheer Kumar, Yi-Yu Liu, Jeremie Lumbroso, Paul Lynch, Richard McKenna, William McQuain, Adam Megacz, Alessandro Montano, Judy Mullins, William Obermeyer, Ivor Page, Gustavo Patino, Christoph von Praun, Klaus Ramelow, David Reimann, Patricia Renault, AndrÃ© Rodrigues, Robert Roos, Joseph Roth, Marco Salinas, Peter Schulthess, Ofer Shaham, Scott Sigman, Sasha Solganik, Timothy Stanley, Gene Stark, Josh Steinhurst, Michelle Strout, Didier Teifreto, Mitchell Theys, Massimo Tivoli, Dwayne Towell, Duy Truong, Judah Veichselfish, Vineeth, Daniel Walker, Janyce Weibe, Ben West, and Armin Zundel.

The Mars.jar file contains all source code and, starting with Release 3.6, the files necessary to generate a new jar file should you wish to make changes to the source and repackage it for your own use. Let us know if you do this, so we can consider your changes for the general release.

Thanks to everyone who uses MARS. Keep those cards and letters coming!

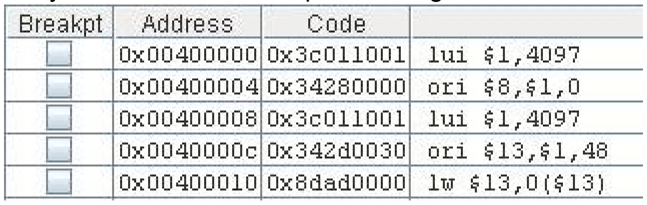
```
# Compute first twelve Fibonacci numbers and put in array, then print
      .data
fibs: .word   0 : 12        # "array" of 12 words to contain fib values
size: .word  12             # size of "array"
      .text
      la   $t0, fibs        # load address of array
      la   $t5, size        # load address of size variable
      lw   $t5, 0($t5)      # load array size
      li   $t2, 1           # 1 is first and second Fib. number
      add.d $f0, $f2, $f4
      sw   $t2, 0($t0)      # F[0] = 1
      sw   $t2, 4($t0)      # F[1] = F[0] = 1
      addi $t1, $t5, -2     # Counter for loop, will execute (size-2) times
loop: lw   $t3, 0($t0)      # Get value from array F[n]
      lw   $t4, 4($t0)      # Get value from array F[n+1]
      add  $t2, $t3, $t4    # $t2 = F[n] + F[n+1]
      sw   $t2, 8($t0)      # Store F[n+2] = F[n] + F[n+1] in array
      addi $t0, $t0, 4      # increment address of Fib. number source
      addi $t1, $t1, -1     # decrement loop counter
      bgtz $t1, loop        # repeat if not finished yet.
      la   $a0, fibs        # first argument for print (array)
      add  $a1, $zero, $t5  # second argument for print (size)
      jal  print            # call print routine.
      li   $v0, 10          # system call for exit
      syscall               # we are out of here.

#########  routine to print the numbers on one line.

      .data
space:.asciiz  " "          # space to insert between numbers
head: .asciiz  "The Fibonacci numbers are:\n"
      .text
print:add  $t0, $zero, $a0  # starting address of array
      add  $t1, $zero, $a1  # initialize loop counter to array size
      la   $a0, head        # load address of print heading
      li   $v0, 4           # specify Print String service
      syscall               # print heading
out:  lw   $a0, 0($t0)      # load fibonacci number for syscall
      li   $v0, 1           # specify Print Integer service
      syscall               # print fibonacci number
      la   $a0, space       # load address of spacer for syscall
      li   $v0, 4           # specify Print String service
      syscall               # output string
      addi $t0, $t0, 4      # increment address
      addi $t1, $t1, -1     # decrement loop counter
      bgtz $t1, out         # repeat if not finished
      jr   $ra              # return
```

MARS Features

This is the overview feature description -- see also new features in each release.

MARS features:

- GUI with point-and-click control and integrated editor. The three icons shown here denote "run", "single-step", and "single-step backwards."
- Integrated editor, featuring multiple file-editing tabs, context-sensitive input, and color-coded assembly syntax. All assembly files in a directory (folder) may be assembled into a single executable.
- Easy set/removal of breakpoints using check boxes



Breakpt	Address	Code	
☐	0x00400000	0x3c011001	lui $1,4097
☐	0x00400004	0x34280000	ori $8,$1,0
☐	0x00400008	0x3c011001	lui $1,4097
☐	0x0040000c	0x342d0030	ori $13,$1,48
☐	0x00400010	0x8dad0000	lw $13,0($13)

- Easily editable register and memory values, similar to a speadsheet
- Display values in hexadecimal or decimal
- Command line mode for instructors to test and evaluate many programs easily. Command-line arguments to specify registers and memory locations to be displayed after the program run, to examine for correct contents. Set up a "batch" to do many programs in succession.
- Floating point registers, coprocessor1 and coprocessor0
- Variable-speed execution

- "Tool" utility for MIPS control of simulated devices (new in 2.1). A tool is a program running on a separate thread with access to MARS data. An assembly program can run in MARS and interact with the tool through memory-mapped IO. Any imaginable pseudo-device can be interfaced to MIPS assembly code, or extended to physical devices or hardware.