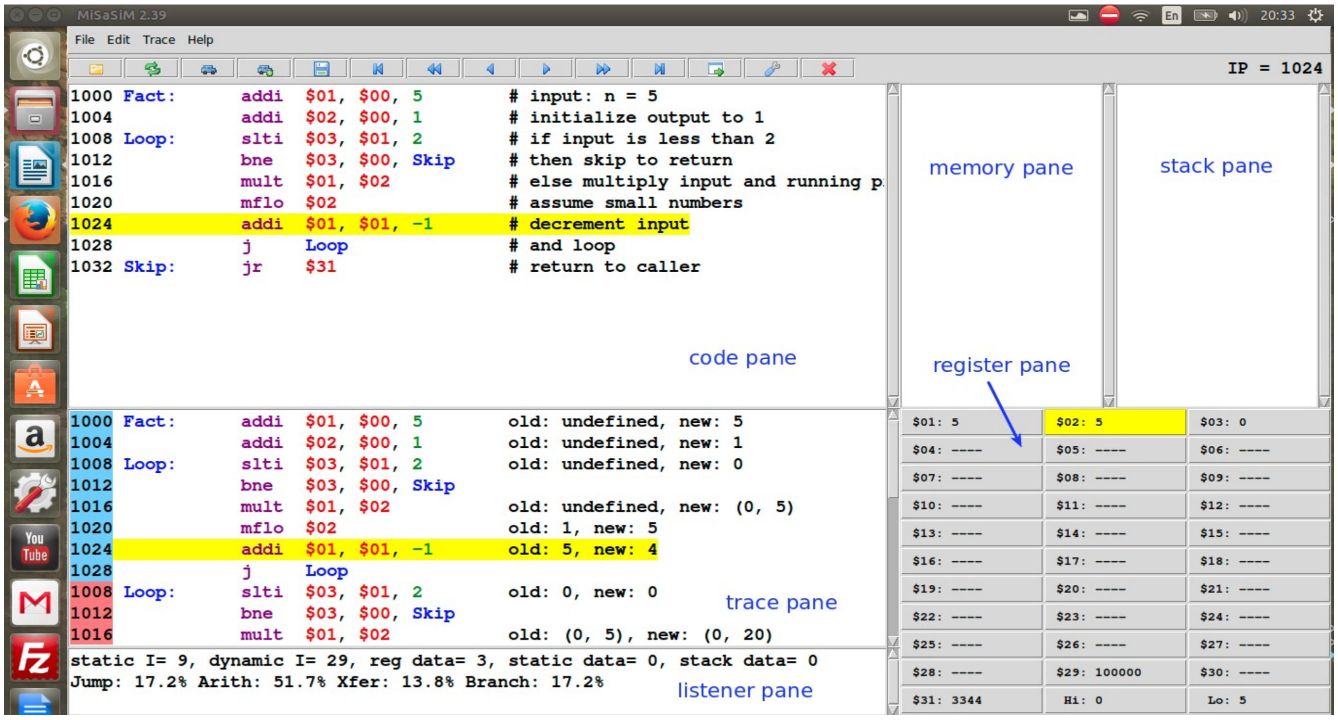
MiSaSiM

A Trace Level Assembly Simulator

MiSaSiM is a simulator for exploration of the functionality and efficiency of MIPS assembly programs. It employs a trace to allow both forward and backward navigation of a program's execution. It also provides operating system calls (through software interrupts) for data initialization, visualizations, etc. It runs under python for maximum platform independence, compact code size, and a rich user interface. This document outlines MiSaSiM's key features and their use. For installation instructions, go to www.ece.gatech.edu/~scotty/misasim/.

When MiSaSiM is launched, the main window opens. In this example, the factorial example "fact.asm" is loaded.



```
1000 Fact:    addi  $01, $00, 5       # input: n = 5
1004          addi  $02, $00, 1       # initialize output to 1
1008 Loop:    slti  $03, $01, 2       # if input is less than 2
1012          bne   $03, $00, Skip    # then skip to return
1016          mult  $01, $02          # else multiply input and running p
1020          mflo  $02               # assume small numbers
1024          addi  $01, $01, -1      # decrement input
1028          j     Loop              # and loop
1032 Skip:    jr    $31               # return to caller
```

```
1000 Fact:    addi  $01, $00, 5       old: undefined, new: 5
1004          addi  $02, $00, 1       old: undefined, new: 1
1008 Loop:    slti  $03, $01, 2       old: undefined, new: 0
1012          bne   $03, $00, Skip
1016          mult  $01, $02          old: undefined, new: (0, 5)
1020          mflo  $02               old: 1, new: 5
1024          addi  $01, $01, -1      old: 5, new: 4
1028          j     Loop
1008 Loop:    slti  $03, $01, 2       old: 0, new: 0
1012          bne   $03, $00, Skip
1016          mult  $01, $02          old: (0, 5), new: (0, 20)
```

```
static I= 9, dynamic I= 29, reg data= 3, static data= 0, stack data= 0
Jump: 17.2% Arith: 51.7% Xfer: 13.8% Branch: 17.2%
```

$01: 5	$02: 5	$03: 0
$04: ----	$05: ----	$06: ----
$07: ----	$08: ----	$09: ----
$10: ----	$11: ----	$12: ----
$13: ----	$14: ----	$15: ----
$16: ----	$17: ----	$18: ----
$19: ----	$20: ----	$21: ----
$22: ----	$23: ----	$24: ----
$25: ----	$26: ----	$27: ----
$28: ----	$29: 100000	$30: ----
$31: 3344	Hi: 0	Lo: 5

Window Panes

The main window is segmented into the following panes.

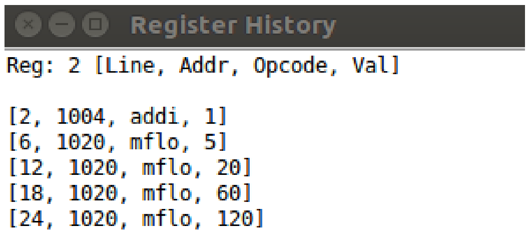
Code pane: displays the source code that is being simulated, with the instruction address shown to the left of each instruction.

Trace pane: displays the series of instructions that have been executed and the changes in the value of each instruction's destination register (if it has one). As the user navigates through the trace, the yellow highlighted line shows the current instruction in the trace pane and the corresponding instruction in the code pane. The instruction addresses are also displayed to the left of the instructions, with color coding to indicate changes in control flow (e.g., at loop body or subroutine boundaries or when branches are taken).

Memory pane: shows the current state (Address: Value) of data words in static or heap memory that have been written so far.

Stack pane: shows the current state (Address: Value) of data words in stack memory. When the stack is popped, the popped values are grayed out.

Register pane: shows the current values in the register file, with the most recently changed value highlighted in yellow. Each register can be clicked to create a box that shows the value changes throughout the program.



```
Reg: 2 [Line, Addr, Opcode, Val]

[2, 1004, addi, 1]
[6, 1020, mflo, 5]
[12, 1020, mflo, 20]
[18, 1020, mflo, 60]
[24, 1020, mflo, 120]
```

The box shows which line in the trace the value changed, the instruction address that caused the value change, the instruction itself, and the value the register changed to.

Listener pane: shows statistics on the executed code, including static instruction count, dynamic instruction count, number of registers used, memory usage, and dynamic instruction mix (i.e., what percentages of different types of instructions occurred in the trace). This pane also shows error and warning messages.

Command Buttons

The command buttons have the following functions.

Load: load a MIPS assembly file.

Reload: reload the currently loaded file. Since the source file is typically edited using notepad, emacs, etc., this command loads the current file version.

Execute: execute the loaded assembly file generating a new trace for instructions executed. This command is automatically executed if the navigation buttons are selected before execution has taken place.

MultiExec: This command executes the loaded program multiple times. The number of executions is user specified. This is useful for computing an average execution time.

Dump: This command dumps the current memory map into a user selected text file. This file contains both memory address and data pairs. Note that the memory values recoded are for the selected trace position.

Start: This command positions the trace at the initial executed instruction (i.e., the trace start).

Prev: This command positions the trace at the last encountered occurrence of the current instruction. This is useful for stepping back to a previous iteration of a loop.

Backward: This command moves backwards in the trace to the instruction executed immediately prior to the current instruction.

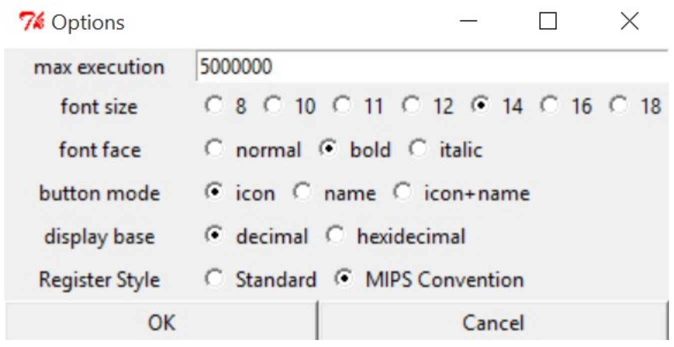
Forward: This command moves forwards in the trace to the instruction executed immediately after to the current instruction.

Next: This command positions the trace at the next encountered occurrence of the current instruction. This is useful for stepping forward to the next iteration of a loop.

End: This command positions the trace at the final executed instruction (i.e., the trace end).

Options: This menu allows the adjustment of:

> Max Execution – The maximum amount of instructions that can be executed. This upper limit value prevents simulator lockup when infinite loops are encountered.
>
> Font Size – changes font size
>
> Font Face – changes the font style
>
> Button Mode – changes the button format from text to icon or both
>
> Display Base – changes display of values from decimal base to hexadecimal base
>
> Register Style – changes the display of registers from standard to the MIPS convention often used in procedure calling

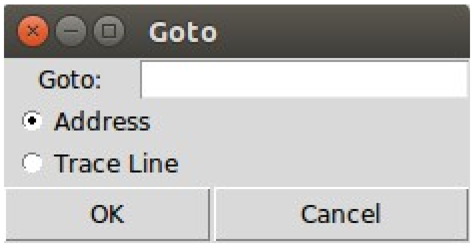


max execution	5000000
font size	8 10 11 12 [✓ 14] 16 18
font face	normal [✓ bold] italic
button mode	[✓ icon] name icon+name
display base	[✓ decimal] hexidecimal
Register Style	Standard [✓ MIPS Convention]

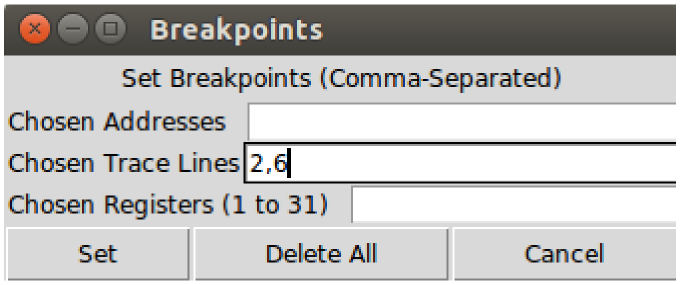
Exit: this command exits MiSaSiM.

Debug Menu

Goto: This command can either jump to an address or a trace line. If an address is selected, the program will go jump either forward or backwards to reach the specified address. If the trace line is selected, goto will jump to the exact point in the trace specified. You can find trace lines in the first column of the register history.

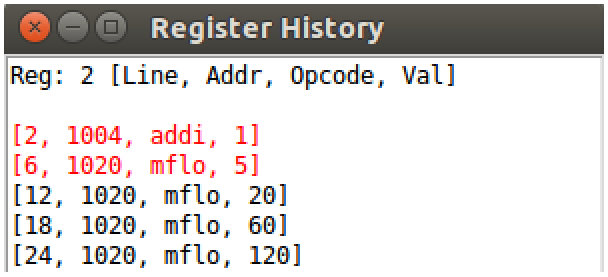


Breakpoints: This command allows breakpoints to be set by either specifying instruction addresses, trace lines, or a register. The values must be comma separated and the breakpoints can be set in any combination of the three types. For example, if given the input: '1012,1024' the simulator will highlight the instructions 'bne $03, $00, Skip' and 'addi $01, $01, -1' red. The Next and Prev buttons will then jump to the breakpoints instead of the default functionality where it jumps to the next occurrence of the current instruction address.



Chosen Addresses
Chosen Trace Lines 2,6
Chosen Registers (1 to 31)

If a breakpoint is set on a trace line or a register, the register history box will show the individual instructions in red.



```
Reg: 2 [Line, Addr, Opcode, Val]

[2, 1004, addi, 1]
[6, 1020, mflo, 5]
[12, 1020, mflo, 20]
[18, 1020, mflo, 60]
[24, 1020, mflo, 120]
```