How to guide and suggested tasks

Thanks for using Simulizer! Simulizer has a lot of varied functionality so this document will hopefully introduce you to the main features you need to complete whatever task you choose to attempt.

Basic Workflow

The toolbar controls most of the workflow. Use the File menu to open and save program files. Use the simulation menu to set the clock speed and control the running of the simulation. There are a lot of different components which make up Simulizer so we have created several different layouts which are easy to switch between using the 'layouts' toolbar.

The editor will notify you of any errors with your program as you type. If you need information about a particular instruction or `syscall`, use the help menu.

Description of Components

The following is a description of the components available in Simulizer:

- **Editor**: The text editor. This is where you write your programs. When you run a program, the line currently being executed will be highlighted (3 lines will be highlighted when using the pipelined version, one for each stage)
- **CPU Visualisation**: A visualisation of data moving through the CPU while running! This is Simulizer's main visual component and aims to help you understand what happens when you run your program. Hover over components for tooltips. (Make sure to set the clock speed low (0.5 or lower) to be able to read what is going on)
- **Options**: A window to configure Simulizer to your liking.
- **High Level Visualisation**: A component to reinforce your knowledge of algorithms. With the packaged programs, if you run an algorithm, you can enable a visualisation showing the algorithm being visualised. Currently supported visualisations are: tower of Hanoi and list/array visualisation. If you want to enable visualisations on your own algorithms, you can do! Just annotate your code using the JavaScript annotation system (info about which is in a separate document).
- **Labels**: Displays all the labels used in your program.
- **Program I/O**: Your main source of I/O while running a program. There are 3 streams available to you: standard, error and debug.
- **Memory View**: A window which allows you to analyse the current state of memory while your program is running.
- **Registers**: A window to display the contents of all of the general purpose registers while your program is running.

General Tips

Here are some general tips for you while using Simulizer:

- **Debugging**: Everyone loves adding print lines into you high level program to check it's doing something, right? Well in MIPS, a simple task like this can take a few lines and possibly even mess your code. Instead, why not use the JavaScript annotation system to put those statements in instead! It's as easy as `# @{ log('pie!') }@` (see the annotation guide for more)
- **Viewing the simulation**: Everything in Simulizer is synced up to each other, therefore if you set a really high clock speed, your program will run really fast, but all the visualisations will run really fast too! If you want to view the visualisations fully, it is suggested to use a relatively low clock speed (0.5 - 4 is a sensible range (but feel free to go crazy too))
- **Shortcuts**: If you don't like clicking things, certain functions in Simulizer have shortcut keys. These can be viewed in the toolbar. Additionally, if you like vim, then you can toggle the editor to vim mode in the options menu.
- **Prototyping**: Writing directly to a low level language can be challenging. By prototyping sections of code in javascript, you can get your algorithm working, then convert the javascript sections to SIMP. Try using the `get` and `set` methods of the registers inside annotations: `# @{ $s0.set($s0.get() + 15 + ($s1.get() * 2)) }@`

Suggested Testing Tasks

If you've got to this stage in the document (or have just skipped the rest), thanks again for your willingness to Simulizer! Feel free to do whatever you want in you use of Simulizer but if you are stuck for ideas, here are some tasks you could try:

- Try running the tower of Hanoi code which should be on default on load-up and watch the high-level visualisation.
- Write a trivial MIPS program, but try and use the annotation system to print a message to the console.
- If you're feeling daring, why not try and run your binary search submission for Computer Systems & Architecture on Simulizer and see what happens?
- Run bubble sort and set the clock speed at an appropriate speed to be able to watch the CPU visualisation (`Windows>CPU Visualisation`).
- As a bit of fun, why not try writing a program using the following two mysterious `syscall` codes `67697865`, `82736775`... (Hint: These are ASCII codes)

Bugs

Obviously, Simulizer is still in development, and so will probably still contain a million and one bugs. Whilst we try to find as many of these bugs as possible, we can't guarantee to find them all. Therefore, if you find a bug during your use of Simulizer, regardless of how small or big, we want to know! Any bugs you do find, please add to the testing questionnaire. (and please include a description of how you broke it, and a stack trace if you get one)

Questionnaire

Finally, once you have finished using Simulizer (or got so annoyed with it you threw your laptop out of your window), please could you fill in the questionnaire at http://goo.gl/forms/qAVSjpWgYw. It shouldn't take more than 10 minutes and would mean a great deal to us if you could fill this in honestly. Thanks again!