Ripes Introduction

Ripes is a graphical processor simulator and assembly code editor built for the RISC-V instruction set architecture, suitable for teaching how assembly level code is executed on various microarchitectures.

The following sections serve as an introduction to the main features of Ripes.

The Editor Tab

The editor tab shows two code segments. On the left hand side, it is possible to write an assembly program written using the RISC-V RV32(I/M) instruction sets. Whenever any edits are performed in this assembly program - and no syntax errors are found - the assembly code will automatically be assembled and inserted into the simulator. If a C compiler has been registered, the `input type` may be set to `C`. It is then possible to write, compile and execute C-language programs within Ripes, see

this wiki page for more detail.

Next, on the right hand side a second code view is displayed. This is a non-interactive view of the current program in its assembled state, denoted as the *program viewer*. We may view the assembled program as either disassembled RISC-V instructions, or as the raw binary code. The blue sidebar of the right-hand view may be clicked on to set a breakpoint at the desired address. Pressing the icon will bring up a list of all symbols in the current program. Through this, it is possible to navigate the program viewer to any of these symbols.

Ripes is bundled with various examples of RISC-V assembly programs, which can be found under the `File->Load Examples` menu.

An example program could be the following: loading a value from memory into a register and incrementing the value.

```
.data
w: .word 0x1234

.text
lw   a0 w
addi a0 a0 1
```

With a program ready to be simulated, we may now move to the *Processor tab*.

The Processor Tab

The processor tab is where Ripes displays its view of the currently selected processor, as well as any additional information relevant to the execution. Apart from the processor view, the processor tab contains the following views:

- 1: **Registers**: A list of all registers of the processor. Register values may be **edited** through clicking on the value of the given register. Editing a register value is immediately reflected in the processor circuit. The most recently modified register is highlighted with a yellow background.
- 2:Instruction memory

: A view into the current program loaded in the simulator.
    - **BP**: Breakpoints, click to toggle. Any breakpoint set in the editor tab will be reflected here.
    - **PC**: The address of the given instruction
    - **Stage**: Lists the stage(s) that is currently executing the given instruction
    - **Instruction**: Disassembled instruction
- 3: **Statistics**: Various statistics based on the cycle count and current number of retired instructions.
- 4: **Output**: Any output through an `ecall` print function will be displayed here.

The Processor View

Processor models in Ripes communicate the current state of the datapath through various visual means, such as

- Multiplexers indicate the currently selected input signal by highlighting an input port with a green dot.
- Various components contains indicators to communicate whenever i.e. a register is clocked, a branch is taken, etc.
- Port value changes are reflected through signal wires:
    - *Boolean (1-bit signals)*: Boolean signals will always indicate whether a signal is high (1) when a wire is green, and low (0) when a wire is grey.
    - Other signals, when modified, will briefly flash green to indicate that the value was modified.

The processor view may be zoomed by performing a **ctrl+scroll** operation (cmd+scroll on OSX).

Clicking a wire highlights the entirety of the wire. This is useful when trying to deduce how a signal is routed through the datapath in some of the more complex layouts. Given that Ripes simulates the entire datapath of a processor, it is possible to investigate the value of any signal, at any point in time.

1. Hover over any port in the processor view. This will display the name of the port, as well as the current value of the port.

1. Press the *Display signal values* button. This will display the output values of all output ports in the design. Alternatively, right click on any port and press "*show value*" to display its label. If a port's value label has been made visible, it is possible to change the radix of the displayed value through right-clicking the port label.

Controlling the Simulator

The toolbar within Ripes contains all of the relevant actions for controlling the simulator.

Select Processor	**Reset**	**Reverse**	**Clock**	**Auto-clock**	**Run**	**Display signal values**	**Show stage table**

- **Select Processor**: Opens the processor selection dialog (for details, refer to section below).
- **Reset**: Resets the processor, setting the program counter to the entry point of the current program, and resets the simulator memory.

- **Reverse**: Undo's a clock-cycle.
- **Clock**: Clocks all memory elements in the circuit and updates the state of the circuit.
- **Auto-clock**: Clocks the circuit with the given frequency specified by the auto-clock interval. Auto-clocking will **stop** once a breakpoint is hit.
- **Run**: Executes the simulator **without** performing GUI updates, to be as fast as possible. Any print `ecall` functions will still be printed to the output console. Running will **stop** once a breakpoint is hit or an exit `ecall` has been performed.
- **Display signal values**: Toggles displaying all output port values of the processor.
- **Show stage table**: Displays a chart showing which instructions resided in which pipeline stage(s) for each cycle. Stalled stages are indicated with a '-' value. **Note**: Stage information is *not* recorded while executing the processor through the *Run* option.

While executing the program loaded earlier, we may observe that, in cycle 4, a load-use dependency arises between the 2nd and 3rd instruction. This results in the `ID` stage being stalled for one clock cycle, whilst the load is being performed. Pipeline stalls (due to hazards) and flushes (due to control flow) will be indicated above a pipeline stage as `nop` instructions highlighted in red.

Selecting Processor Models

Through providing multiple processor models, Ripes provides the ability to investigate how different microarchitectures affect program execution. The set of processor models shipping in version 2.0 (described below) aims to address each level of added complexity when going from a single cycle processor to a fully functioning, in-order pipelined processor. Ripes provides the following processor models:

- **RISC-V Single Cycle Processor**
- **RISC-V 5-Stage Processor w/o Forwarding or Hazard Detection**
- **RISC-V 5-Stage Processor w/o Hazard Detection**
- **RISC-V 5-Stage Processor**

Furthermore, each processor provides multiple layouts of the processor. By default, the following two layouts are provided:

- **Standard**: A simplified view of the processor. Control components and signals are omitted.
- **Extended**: An extended view of the processor. Control components and signals are visible as well as wire bit-widths.

Opening the processor selection dialog, one may choose and configure the current processor:

On the left hand side, each available processor is listed. As for configuration, a layout may be selected from the list of layouts. Note that the layout does **not** affect the inner workings of the processor model, only the displayed components. Finally, it is possible to specify register initializations. These initialization values will be applied to the registers each time the processor is reset.

As an example processor selection, the following image shows the *extended* layout of the *RISC-V 5-stage Processor*:

The Memory Tab

The memory tab provides a view into the entire addressable address space of the processor, as well as access to Ripes' cache simulator.
Navigating the memory may be done as follows:

- **Scrolling** the memory view

- **Go to register** will scroll the memory view to the value currently present in the selected register
- **Go to section** will scroll the memory view to the address of the given section value in memory (i.e. the instruction memory `.text` segment, the static data `.data` segment etc). Furthermore, a custom address may be specified through the "Address..." option.

Whenever the processor is reset, all memory written during program execution will be reset to its initial state.