Visualiser Documentation

Ruben Saunders

November 2024

Contents

1 Overview

The visualiser offers an interactive view into code execution and the interaction between the compiler, the assembler, and the processor. The visualiser

- Is machine-code executor, able to run the loaded binary on the processor in a step-wise fashion.
- Has a register view/editor.
- Has a memory view/editor.
- Is a basic debugger.
- Shows the current machine-code instruction, and back-traces it to both the assembly and high-level source code.

The application is a TUI (a textual user interface), meaning it runs in the terminal. The interface is tab-based, with a navigation bar along the top. Later sections will cover the content of each tab.

1.1 Command-Line Interface

The visualiser executable is called as follows:

```
$ ./visualiser [filename] [flags]
```

If a filename is provided, this is assumed to be the root name of all components: `filename.edel`, `filename.asm`, `filename.s`, and `filename` (binary). Each file may be provided using a flag which will override the assumption:

- `--edel <file>` – sets the input high-level Edel source file. Default: `<filename>.edel`.
- `--asm <file>` – sets the input assembly source file. Expects output from the compiler. Default: `<filename>.asm`.
- `--reconstructed <file>` – sets the reconstructed assembly file. Expects `-r ...` file from the assembler. Default: `<filename>.s`.
- `--bin <file>` – sets the binary file. Expects output from the assembler. Default: `<filename>`.

There are some additional flags available:

- `-b <list>` or `--breakpoints <list>` – a comma-separated list of $pc values to install breakpoints at.
- `--stdout <file>` – pipe the processor's output to this file.
- `--stdin <file>` – set the processor's input stream to this file.

1.2 Execution Example

Below is the sequence of steps to go from a source Edel file to running it on the visualiser.

1. Write a high-level source file, say `source.edel`.
2. Compile the file, emitting debug comments. The following command will output an assembly file, `source.asm`.

   ```
   $ ./compiler source.edel -o source.asm -d
   ```

3. Assemble the file, emitting a reconstruction. The following command will output a binary, `source`, and a reconstruction mapping, `source.s`.

   ```
   $ ./assembler source.asm -o source -r source.s
   ```

4. Launch the visualiser. If all the file names are the same, with the correct extensions, we can simply pass the file name without an extension.

   ```
   $ ./visualiser source
   ```

1.3 Navigation & Control Basics

As a TUI, everything may be done using keyboard controls. Navigation between key elements is done with the arrow keys (e.g., navigating between tabs, selecting registers in the list).
To use the tab-specific controls, the tab body must be focused. This can be achieved by pressing the down arrow. Key controls are detailed in this document; however, a summary of controls for each tab are listed at the bottom of the screen.
Aside from tab-specific controls, the following are global:

- `Ctrl+c` – exits the application.
- `F`n – selects the n^{th} tab.

2 Code Execution

The "main" tab, this view shows the source and compiler assembly side-by-side.

2.1 Layout

At the top of the screen are two checkboxes, one which toggles line selection, and the other which determines if the processor is currently running. Below this is the source code of the processor, split into the following panes:

- The assembly source is displayed on the left, with its file name displayed.
- The reconstructed compiler assembly is display on the right.

The current line pointed to by $pc is highlighted in yellow, which is traced back to the source assembly. If enabled, the currently selected line is displayed in cyan, with the corresponding equivalent lines highlighted in light cyan in the other pane. Breakpoints are indicated by a • preceding the line.
Below this is a section for the processor's debug messages, which is hidden by default unless messages are available. The fields below this display the

- Processor's status – either *halted* or *running*. This reflects the `is_running` bit in $flag.
- Cycle – indicates the processor's current cycle, i.e., number of executed steps.
- Program counter – the current value of $pc.

2.2 Keyboard Controls

The pane's displaying file contents behave as scrollable windows. They may be navigated by the arrow keys as well as by the scroll wheel.

- `Return` – commends the fetch-execute cycle, executes until a breakpoint is encountered or the program halts.
- `Space` – executes the current line. In the machine code pane, this is equivalent to a single CPU cycle.
- `b` – toggles a breakpoint for the selected line.
- `h` – toggles the `is_running` bit in $flag.
- `j` – if line selection is enabled, sets $pc to the first selected line in the compiled assembly.
- `r` – resets $flag: sets `is_running` and clears any error bits.
- `s` – toggle line selection.

3 Registers

This view shows a list of the processor's registers.

3.1 Layout

The left-hand side of the view displays a list of the processor's registers, accompanied by their 64-bit hexadecimal value. This list is divided into two groups: special registers, and general purpose registers. When a register is selected, more information about that register is display in a pane on the right. This pane lists the register's value in hexadecimal, as a decimal integer (32-bit) and long (64-bit), as a float, and as a double. Each may be edited and updated. A brief description of the register is given below this.
The $flag register has an additional section below this. $flag contains state information about the processor, which are listed in a decomposed form here.

3.2 Keyboard Controls

When an input field is selected, controls work as they ordinarily would in an input box, overriding the below.

- `1-9` – select register `$r`n.
- `Backspace/Delete` – zeroes the current register.
- `c` – copies the contents of the current register.
- `r` – refresh all registers, and the selected register's pane. Useful when a register value has been updated in another tab, and changes have not been loaded.
- `Up/Down arrows` – scroll in the register list.
- `v` – paste the copied register's value into the current register.

4 Memory

This view displays a section of memory in a grid view.

4.1 Layout

The main component in this tab is the grid of memory cells. Violet values along the side indicate the base address of each row, with the values along the top marking offsets. The range of addresses shown in this grid are indicates in the title.
Once the grid is focused, the current address will be highlighted. Moving this cursor will be detailed in the next section, but note that if the user tries to move the cursor beyond the view's borders, the view will attempt to scroll if possible. Additionally, the value of $pc is highlighted in yellow.
Below the grid is a dropdown and textbox, used for viewing and editing memory more than one byte at a time. The dropdown dictates the datatype of the data being read. This may be changed by focusing the dropdown and pressing `Enter`, then using the arrow keys to navigate the list, and pressing `Enter` again to select the value. The textbox next to this displays data of this type. If this section is focused, the bytes being read are highlighted in the memory view.

4.2 Keyboard Controls

These controls only take effect when the memory grid is in focus.

- `[0-9a-zA-Z]` – modifies the byte in memory. Specifically, the lower four bits are overwritten with this value, while the old bytes are shifted into the upper half.
- Arrow keys – navigate the memory view, moving the cursor in the given direction.
- `Backspace/Delete` – zeroes the current byte.
- `Ctrl+PageUp/Down` – navigates to the absolute start/end of the processor's memory region.
- `Enter` – move focus to the textbox.
- `Home/End` – navigate to the start/end of the current line.
- `PageUp/Down` – navigates to the start/end of the memory view.
- `Tab` – move focus to the datatype dropdown.

5 Sources

This tab displays all the source files used in the program.

5.1 Layout

The pane on the left categorises the files into which source layer they fit into: machine code (the reconstructed assembly provided via `--asm`); assembly sources (files inputted to the assembler); and high-level source files (files inputted to the compiler). The list may be navigated to select files, wherein there contents are displayed on the right-hand pane. The number of breakpoints in a file, if any, are indicated by $n\times$● preceding the filepath.
The right-hand pane displays the contents of the selected file. The pane may be selected and arrow keys used to change which line is selected (the line highlighted in blue). The line containing the current $pc is also highlighted in yellow. Breakpoints are indicated by a ● preceding the line.

5.2 Keyboard Controls

- `e` – changes the file and line selection to point to the location of the current $pc.

The following controls only take effect when the file pane is in focus.

- `[` and `]` – both change the file and line selection by tracing the selected line forwards or backwards, respectively. That is, if a source (`.s`) line is selected, `[` will select the line in an assembly (`.asm`) source, whereas `]` does nothing.
- `b` – toggle the breakpoint on the selected line.
- `j` – jumps to the current line, i.e., sets $pc to the location of the selected line.

6 Settings

Contains various visualiser settings.

6.1 Layout

The tab contains several bordered windows, grouping common settings together.

Debug Flags Controls the debug flags of the processor. For information, see the `-d...` flags in the processor documentation.