SigmaEmu Design Document

Jake Lawrence

March 24, 2023

This document was written to document the various parts of the design of the SigmaEmu app, so that future developers can work on it more seamlessly. It is assumed the reader already knows what SigmaEmu is and has a decent knowledge of Sigma16.

1 Architecture

The SigmaEmu project was broken down into three assemblies (effectively the C# equivalent of modules, that allows less stuff to have to be compiled when changes are made. See the C# docs for more detail):

- **Assembler**. This is responsible for parsing Sigma16 files into syntax trees using a parser, which are then traversed in order to assemble the program into machine code. It has no knowledge of **Core**
- **Core.** This holds the code for the app itself, which depends on and makes use of the **Assembler** module.
- **Shared**. This holds shared data-objects that **Assembler** and **Core** both need to make use of in order to communicate. It is depended on by both **Core** and **Assembler**.

1.1 The Assembler

The assembler module is split into two further packages: the Parser and the Assembler. The parser contains a grammar definition of the Sigma16 language, which is used with ANTLR in order to generate a Parser.

The Assembler package contains a Listener implementation which builds a `Listing` object via the process of the various methods being called when relevant nodes are encountered when walking through the tree.

1.2 Core

This is a Blazor app which implements the main logic and UI components of the app. The layout is standard for Blazor projects, so refer to the Blazor docs for more info. The Models folder contains representations of Sigma16's virtual

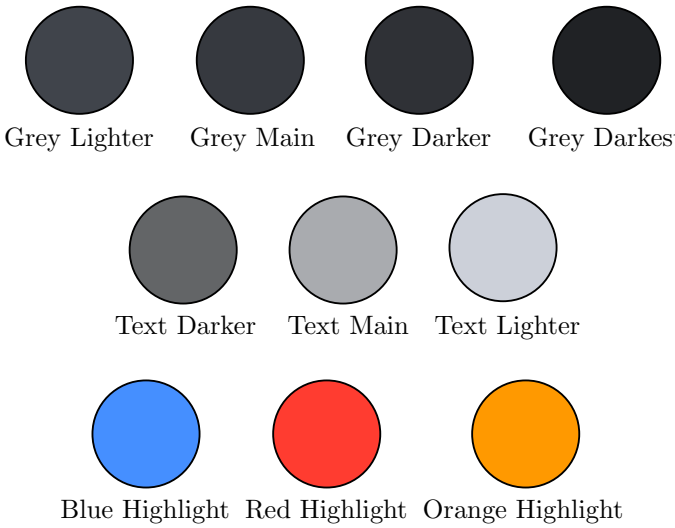

Figure 1: Colour palette used for the design of the interface for SigmaEmu. It primarily uses a range of greys, with bright colours reserved for highlights

hardware, including the processor which is where much of the business logic of the app resides.

1.3 Shared

This houses any shared objects that both the Assembler and Core modules need to understand in order to communicate, such as a `Listing` object.

2 Styling

Initial UI wireframes were designed in Figma and are available here[1]. The wireframes have been the main beacon of design, and have only been strayed from in minor ways.

Figure 1 shows the colour palette used in the design of SigmaEmu. The only colours occasionally used but not defined in this palette are pure white and pure black.

In practice the styling in SigmaEmu is done using a custom extension of bootstrap, compiled using SASS, which allows custom colours and utility classes to be defined. In general, the following principles should be applied when styling new features of SigmaEmu:

[1] `https://www.figma.com/proto/3nA1RdifO4WIOIrlbH4x9I/SimgaEmu-Wireframe?node-id=1-2&starting-point-node-id=1%3A2`, in case there is a rendering issue on the link

- Where possible, do not write any CSS. Instead, try to use utility classes. This style is described further in the docs for Tailwind CSS, for further reference (we are not using tailwind CSS, but the site is a helpful reference)
- All boxes should uses the `rounded-2` class, for rounded edges
- Use the different levels of grey described in figure 1 for distinguish different depths of the interface. Text within a given area should use an appropriate colour to contrast against the background
- Buttons which are inactive should use a darker colour than when active
- Where possible, all buttons should use icons, preferably as well as text.
- Flexbox should be used for laying out data, and for centering items
- Flexbox's new `gap` property should be used for spacing out items, rather than using margins

3 Known Bugs

The following are a list of known bugs, which ideally should be fixed by any future developers

- Footer bar overlaps the bottom of the listing display, making some of it unreadable
- The syntax allows writing multiple instructions on one line, but programs expressed in this way to do assemble properly. Either this should be disallowed or the assembler should be altered to assemble these programs properly, preferably the latter.
- When the screen size is small enough, the sidebar needs a scrollbar, and this messes up the spacing of other elements.
- If one opens a new file very fast (has only been replicated by spam-clicking), the specified properties to not properly load, so the editor is in light-theme.
- Sometimes the editor is still visible after clicking assemble (this was reported by a user during the user study, but has not been able to be replicated. It is unclear whether they meant that *only* the the editor is visible and the listing tab is not there)

4 Potential Further Features

The following is a list of potential features that a future developer could implement / work on. For features marked with a question mark, it should be further considered whether these are valuable features or not:

- Allow pressing control-S to save the opened file. It is likely this can be set up through the monaco component
- Make the editor component resize when the window is resized or zoomed
- Refine the processor controls to make them more intuitive
- Allow users to customise memory size
- Investigate performance issues with larger memory size
- Allow user to toggle whether R0 is forced to be 0 or not?
- Allow user to customise which operations will reset the memory vs which write on top of what's already there
- Allow users to load programs with an offset? (e.g. load program at address 0004, and set PC to 0004)
- Change the syntax errors ot be more human-readable. Currently they are auto-generated by ANTLR's parser.
- Style the scroll-bars on Chrome - currently they do not fit the aesthetic, but do on Firefox
- Allow user to customise whether comments and empty lines should be included in the assembled listing
- Show some kind of message on division by zero, instead of simply halting?
- Add more error handling around case where the assembled code is larger than the digital memory.
- Mousing over hex values shows potential interpretations of them, e.g. as instruction, as signed decimal, etc.
- Mousing over instructions to show documentation on what they do
- Clicking on any hex value scrolls to that address in memory
- Memory shows labels at the relevant address

These are just a few of the features proposed for SigmaEmu but not implemented. Further features could also bring it closer in line with Sigma16 version 3.5.0, for example the addition of an interrupt system.

5 Contact

This app was originally developed by Jake Lawrence, so if you have any further questions or queries, contact me at jakesamuellawrence@gmail.com.