MARINADE

Software Requirements Specification

Prepared by

Curt Henrichs

Hunter Parks

Alex Roberts

Larry Skuse

Andrew Stoehr

Submitted on October 26th, 2017

Milwaukee School of Engineering Senior Design Team Beta
Electrical Engineering and Computer Science Department | Advised by Professor Barnekow

Table of Contents

System Description

MARINADE stands for **M**SOE **AR**M **In**tegrated **A**rchitecture **D**evelopment **E**nvironment. It is designed to be an educational computer architecture simulator for the ARM instruction set. MARINADE is intended to give undergraduate students studying computer architecture insight into how the processor and memory interact with various ARM instructions through visualizations and simulations.

Definitions, Acronyms, and Abbreviations

- MARINADE: MSOE ARM Integrated Architecture Development Environment (often referred to as the *system* in this document)
- ARM: Advanced RISC Machines
- RISC: Reduced Instruction Set Computer
- IDE: Integrated Development Environment (also referred to as the *editor* in this document)
- CE-1921: Computer architecture course
- CE-2801: Entry level embedded systems course

Requirements

The system requirements look at what the system should do. The requirements are broken down into four subsections. The *General Requirements* subsection focuses on the requirements of the system as a whole. The *Editor Requirements* subsection focuses on what functions and features the editor will be required to have. The *Simulator Requirements* subsection addresses what features the simulator will make available to the user. The *Hardware Requirements* subsection looks at what requirements devices running this system must have. It also lists any requirements the system must have to program and external development board.

General Requirements

Description: System must display a text editor to the user.
Rationale: Text editor provides ability for user to develop software within the application.
Priority: Essential

Description: System must display processor simulations.
Rationale: Processor simulations provides users with a deeper look into the ARM architecture.
Priority: Essential

Description: System must show a register view to the user. The register view shows each register and its contents.

Rationale: A register view shows the user how the ARM instruction set interacts with registers.
Priority: Essential

Description: System must show a memory view to the user.
Rationale: A memory view shows the user how the ARM instruction set interacts with memory.
Priority: Essential

Description: System must be easy to install and configure.
Rationale: The system is intended to be used by novice computer engineering students. Users should not have to spend time configuring their environment.
Priority: Essential

Description: System interface must be simple and clean, while not doing a disservice to users.
Rationale: Users should be able to quickly find their way around the program, but they should not feel like they are being insulted with a childish interface.
Priority: Essential

Description: System must meet needs to achieve course outcomes for CE-1921 and CE-2801.
Rationale: System is being developed with the goal of providing a complete development solution for engineering students learning computer architecture and embedded systems.
Priority: Essential

Description: System must have a lifetime of at least 5 years.
Rationale: The system will be able to adapt to changes in the curriculum.
Priority: Essential

Editor Requirements

Description: Editor must provide syntax highlighting.
Rationale: Syntax highlighting provides the user with feedback on the structure of the program making development easier.
Priority: Essential

Description: Editor must read and write text files.
Rationale: Storage of user's application code for future use allows user to write more complex code to test the architecture.
Priority: Essential

Description: Editor must provide standard file manipulation controls (Open, Save, Save as…, Save All, Close).
Rationale: Standard file manipulation controls gives the user a more intuitive environment.
Priority: Essential

Description: Editor must provide notifications for compilation errors.
Rationale: Notifications for compilation errors helps the user immediately identify any compilation errors in their code.
Priority: Essential

Description: Editor must provide notifications for runtime errors.
Rationale: Notifications for runtime errors helps the user immediately identify any runtime errors in their code.
Priority: Essential

Description: Editor must show line numbers.
Rationale: Line numbers can be used to identify the location of errors in a user's code.
Priority: Essential

Description: Editor must provide a file manager for opening and saving project files.
Rationale: User is provided with a file selection dialog box similar to the file selection dialogs found in other development tools process to reduce learning curve.
Priority: Essential

Description: Editor must provide standard refactoring controls (Rename, Delete, Move, Copy, Safe Delete).
Rationale: Standard refactoring controls gives the user a more intuitive environment and makes it easier to make changes to already-written code.
Priority: Essential

Description: Editor must provide the ability to run the ARM simulator with specific configuration options.
Rationale: Integrating the ARM simulator provides cohesive experience when developing and testing software.
Priority: Essential

Description: Editor should provide detailed information about an ARM instruction when the user hovers over it.
Rationale: The detailed information about the ARM instruction will give the user a deeper understanding of the ARM instruction set.
Priority: Essential

Description: Editor should allow different views of the code shown.
Rationale: Viewing the code as it moves through the compilation process will allow users to understand how a computer uses the code that humans can read and write.
Priority: Essential

Simulator Requirements

Description: Simulator must be configurable for multiple processor configurations (single-cycle, multicycle, and pipelined processors).
Rationale: By making various processor configurations available to the user, the user can analyze how different ARM instructions move through different configurations.
Priority: Essential

Description: Simulator must illustrate advanced computer architecture topics.
Rationale: By illustrating more advanced computer architecture topics, the user can better visualize how more advanced computer architecture concepts are implemented in an ARM processor.
Priority: Essential

Description: Simulator must provide a performance analysis tool for comparison of different processor configurations (single-cycle, multicycle, and pipelined processors).
Rationale: A performance analysis tool allows the user to better understand how different processor configurations affect the performance of the system.
Priority: Essential

Description: Simulator must provide a data visualization tool for comparison between different processor configurations (single-cycle, multicycle, and pipelined processors).
Rationale: Data visualization tool allows user to better understand the memory usage differences between processor configurations.
Priority: Essential

Description: Simulator must allow user to step through their code in forward or reverse directions.
Rationale: Step into and step back commands allow the user to visualize ARM instructions moving through the processor at a controlled pace.
Priority: Essential

Description: Simulator must allow user to modify registers, memory, and other components.
Rationale: User has flexibility of manually adjusting values to understand the consequences of that modification.
Priority: Essential

Description: Simulator must highlight changes that occur in hardware between instructions.
Rationale: By highlighting changes that occur, the user's attention is immediately drawn to where the change was made, eliminating the need to search for the change.
Priority: Essential

Description: Simulator must allow for simulated memory mapped input and output devices.
Rationale: User will gain experience working with the full embedded systems model by using simulated hardware devices as input and output of the simulated processor.
Priority: Essential

Description: Simulator must update the register view as soon as register values are changed in the code.
Rationale: User will gain an understanding of when and where data is stored in registers.
Priority: Essential

Description: Simulator must update the memory view as soon as new data is stored in memory by the code.
Rationale: By updating the memory view as soon as new data is stored, the user will see how the memory is manipulated while a program is running.
Priority: Essential

Description: Simulator must provide a dialog box to adjust the settings of the memory view.
Rationale: By providing the dialog box, the user has more control over how they view the memory.
Priority: Essential

Hardware Requirements

Description: The system must be able to run on various computing platforms.
Rationale: Students should be able to run this software on both school provided computers and personal computers.
Priority: Essential

Description: System must support a hardware programmer to interface with physical development boards.
Rationale: User should be able to run developed code on a physical board for educational hardware debugging.
Priority: Time Conditional

Specifications

The system specifications look at how the system will work. The specifications are broken down into four subsections. The *General Specifications* subsection focuses on how the system will be developed whole. The *Editor Specifications* subsection focuses on how the various features of the IDE will look and function. The *Simulator Specifications* subsection addresses how the features of the simulator will look and function. Finally, the *Hardware Specifications* subsection

looks at technical specifications devices running this system must have. It also lists any specifications the system must have to program and external development board.

General Specifications

1. System must make use of existing, open-source technologies.
2. System must support ARMv4.
3. System must convert C and ARM assembly into machine code.
4. System must provide a single package install.
5. System must allow user to perform single-step debugging of both C and ARM assembly code

IDE (Editor) Specifications

1. IDE must provide a tree organizer for selecting projects files.
2. IDE must provide support, such as syntax highlighting, for both ARM assembly and C.
3. IDE must be designed to read and write ARM assembly and C files.
4. Compilation error notifications must provide a clear description of the error.
5. Compilation error notifications must tell the user what line the error occurred on.
6. Runtime error notifications must provide a clear description of the error.
7. Runtime error notifications must tell the user what line the error occurred on.
8. When hovering over an ARM instruction, tooltips must show the ARM instruction machine code.

Simulator Specifications

1. Register view must show all registers and their data using a table that the user can hide.
2. Memory view must neatly layout memory locations in a table.
3. The memory view dialog must allow the word size to be adjusted between 8 bits, 16 bits, and 32 bits.
4. The memory view dialog must allow the range of memory locations displayed to be adjusted.
5. Simulator must be configurable for single-cycle, multicycle, and pipelined implementations of the ARM processor.
6. Simulator must illustrate more advanced computer architecture concepts like hazard management and pipeline stall.
7. The performance analysis tool must show:
    a. Memory pyramid statistics (memory usage, cache hit/miss)
    b. Processor statistics (throughput for all three implementations, number of hazards generated)
8. Simulated input devices must include pushbuttons, switches, general input register.
9. Simulated output devices must include LEDs, 7-segment displays, LCD displays, general output register, and speakers.

10. Device registers will be bidirectional, input only, or output only.
    a. Bidirectional IO registers will have corresponding direction registers.
    b. IDE will allow for user clickable direction and value selection.
11. Simulator will provide a simulated terminal for standard input and output stream behavior.

Hardware Specifications

1. The application must run on Windows 10, macOS, and Linux.
2. (Optional) External development boards must have an ARM Cortex-M or ARM Cortex-R processor.
3. (Optional) External development boards must use JTAG interface for programming and debugging.

Use Cases

The primary use cases for the system are listed below.

1. Creating a new project
    a. The user clicks a “Create Project” button
    b. The user is provided with an interface to enter project details
        i. Required details include a project name, project directory, selection of assembly language or C.
        ii. Optional details include build settings, compiler options, etc. Defaults will be used if optional settings are not specified.
    c. The editor creates a project structure with the user-specified settings.
    d. The editor loads the newly created project and will provide the user with an editing interface.
    e. The editor adds the new project to a list of recent projects.
2. Loading an existing project from recents
    a. The user clicks a “Recent Projects” button
    b. The editor provides an interface listing the user’s recently edited projects
    c. The user clicks the project they would like to work on
    d. The editor loads the project and provide the user with an editing interface
3. Loading an existing project from file
    a. The user clicks an “Open…” button
    b. The editor provides a file dialog interface starting in the default project directory
    c. The user browses the file tree and selects the enclosing folder or project file
    d. The editor loads the project and provide the user with an editing interface
    e. The editor adds the project to a list of recent projects
4. Compiling ARMv4 assembly or C code
    a. The user clicks a “Compile” button

   b. A dialog box pops up asking the user if they would like to save any unsaved code
   c. The system compiles the code using the default compiler settings if no other compiler options have been specified
5. Running ARMv4 assembly or C code
   a. The user clicks a “Run” button
   b. A dialog box pops up asking the user if they would like to save any unsaved code
   c. If the code has not already been compiled, the system compiles the code using the default compiler settings if no other compiler options have been specified
6. Debugging ARMv4 assembly or C code
   a. The user clicks a “Debug” button
   b. A dialog box pops up asking the user if they would like to save any unsaved code
   c. If the code has not already been compiled, the system compiles the code using the default compiler settings if no other compiler options have been specified
   d. The system stops running the code at user defined break points and allows the user to step through, step over, or resume
7. Viewing execution of a loaded project in a single-cycle simulator
   a. The user clicks a “Simulate” button
   b. The editor switches to a “Simulation” view with architecture selection options
   c. The user clicks a “Single-Cycle” button
   d. The editor displays an overview of a simulated processor with single-cycle architecture, including all of the major components of the processor
   e. The user clicks a “Run” button
   f. The editor executes the first clock cycle from the loaded program and updates the display
   g. The user clicks a “Continue” or “Step” button
   h. The editor executes one clock cycle at a time until the end of the program is reached or the user cancels the simulation using a “Cancel” button
   i. At the end of program execution, the IDE displays an interface with “Restart” and “Close” buttons
8. Viewing execution of a loaded project in a multi-cycle simulator
   a. The user clicks a “Simulate” button
   b. The editor switches to a “Simulation” view with architecture selection options
   c. The user clicks a “Multi-Cycle” button
   d. The editor displays an overview of a simulated processor with multi-cycle architecture, including all of the major components of the processor
   e. The user clicks a “Run” button
   f. The editor executes the first clock cycle from the loaded program and updates the display
   g. The user clicks a “Continue” or “Step” button

    h. The editor executes one clock cycle at a time until the end of the program is reached or the user cancels the simulation using a “Cancel” button
    i. At the end of program execution, the editor displays an interface with “Restart” and “Close” buttons
9. Viewing execution of a loaded project in a pipelined simulator
    a. The user clicks a “Simulate” button
    b. The editor switches to a “Simulation” view with architecture selection options
    c. The user clicks a “Pipelined” button
    d. The editor displays an overview of a simulated processor with pipelined architecture, including all of the major components of the processor
    e. The user clicks a “Run” button
    f. The editor executes one clock cycle from the loaded program and updates the display
    g. The user clicks a “Continue” or “Step” button
    h. When pipeline hazards or stalling occur, the editor will highlight these events
    i. The editor executes the first clock cycle at a time until the end of the program is reached or the user cancels the simulation using a “Cancel” button
    j. At the end of program execution, the editor displays an interface with “Restart” and “Close” buttons
10. Viewing performance comparisons for single-cycle, multicycle, and pipelined program execution
    a. User clicks an “Analyze” button
    b. A dialog box pops up
    c. The dialog box shows the number of instructions executed per cycle
    d. The dialog box shows how long the program took to run through each of the various types of processors
11. Simulating input devices
    a. The user selects input device from a menu
    b. The selected device appears on the screen
    c. User can interact with the input state by clicking on the input device shown on the screen
    d. The input device will then respond either by a direct state change or in case of more complex behavior, open a window to configure the device parameters or manipulate device state
    e. If the user writes code that interacts with the visible input device, then the changes initiated by the user will be reflected in the processor simulation
12. Simulating output devices
    a. The user selects output devices from a menu
    b. The selected output devices appear on the screen
    c. If the user writes code to manipulate a visible output device, the changes are reflected on the simulated output device
    d. If user clicks on the device then the device will respond by opening a window to configure the device parameters or manipulate device state

User Interface Mockup

Shown below is the user interface prototype developed to capture the interpretation of the requirements and specifications. The user interface draws heavily upon the technology investigation's research on ARMsim and Visual editors. Additionally, a meeting with the stakeholders influenced the visual presentation by bringing up the multiple views approach. Multiple views allow for the ability to work with a focused view of the workspace so that the actions of the editor mesh with the actions of various levels of hardware simulation in both a visual and logical manner such that it benefits learning for the student. The following image, shown in Figure 1, displays the first view that a student will interact with, the editor. The goal is to present a development environment that is simple but powerful enough to write C and assembly applications. The editor will also be the mechanism to inspect the compilation process for the simulator, and will provide tooltips to view assembly or machine code for a given instruction. Figure 2 presents the memory view which will show changes in the registers' values as the program is simulated. As the program runs, the student can track the changes in the registers for debugging and verification of their programs. The simulator is shown in Figure 3, where the student will interact with the selected ARM processor to develop an understanding and appreciation of the architecture. Both the processor architecture and the peripherals connected to it are shown. Finally, Figure 4 shows the menu used to switch the active view. The student will use this context switch to efficiently switch focus between programming, verification and debugging, and processor dataflow.

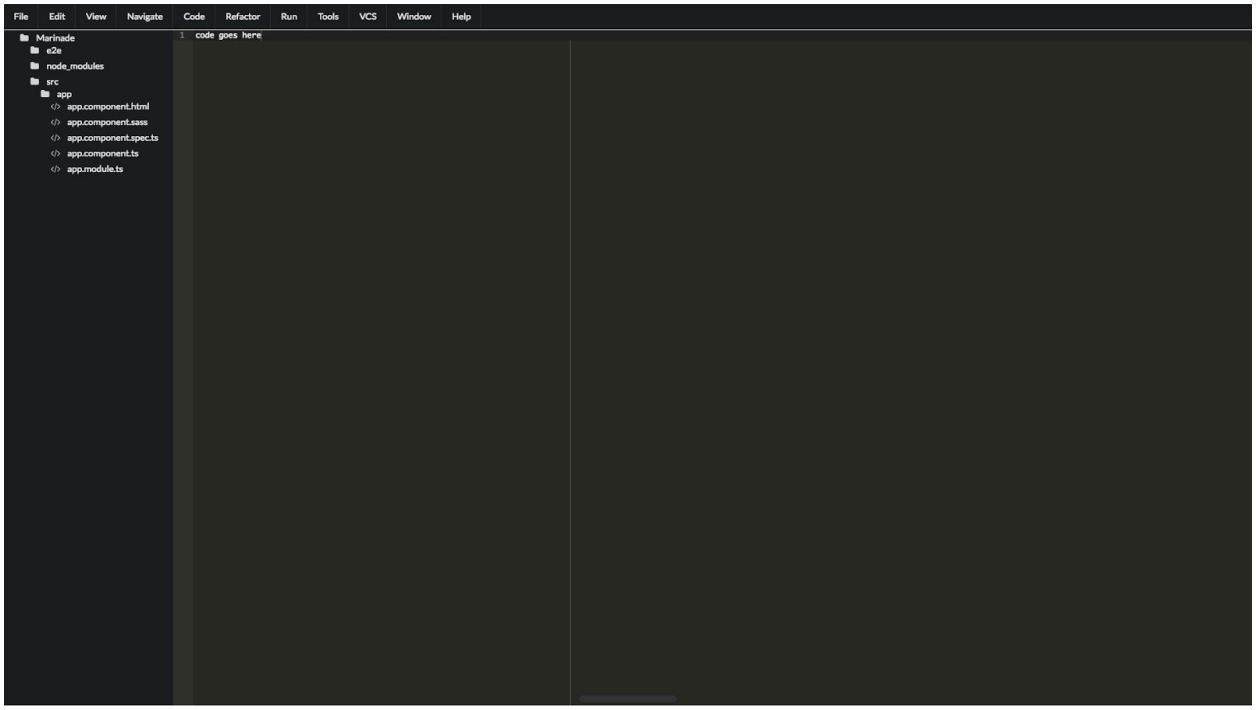

Figure 1: The editor view. This view is where code can be edited and will be the view used most during normal project development.

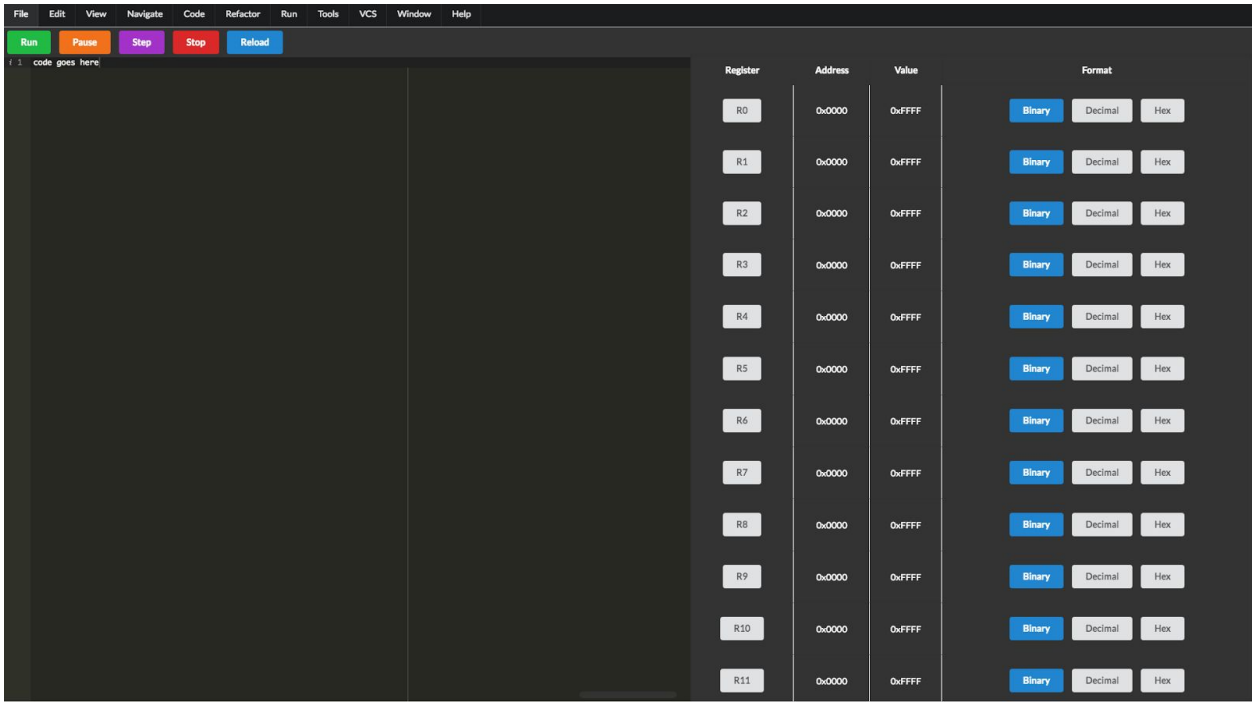

Figure 2: The memory view. This view allows users to step through their code and track register values. This view allows users to select whether to view values in binary, decimal, or hexadecimal for each register individually.

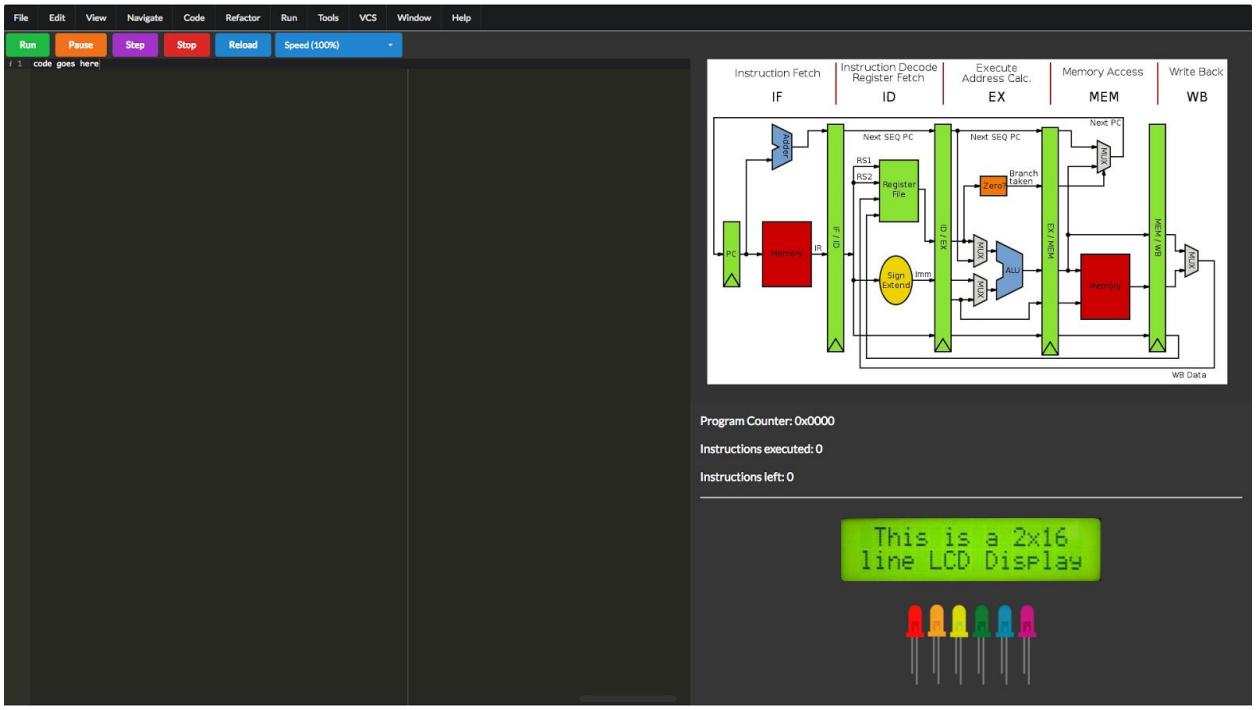

Figure 3: The simulator view. This view allows users to visually track where data is flowing through the processor. This view also includes simulated peripherals like LCD panels, LEDs, and more.

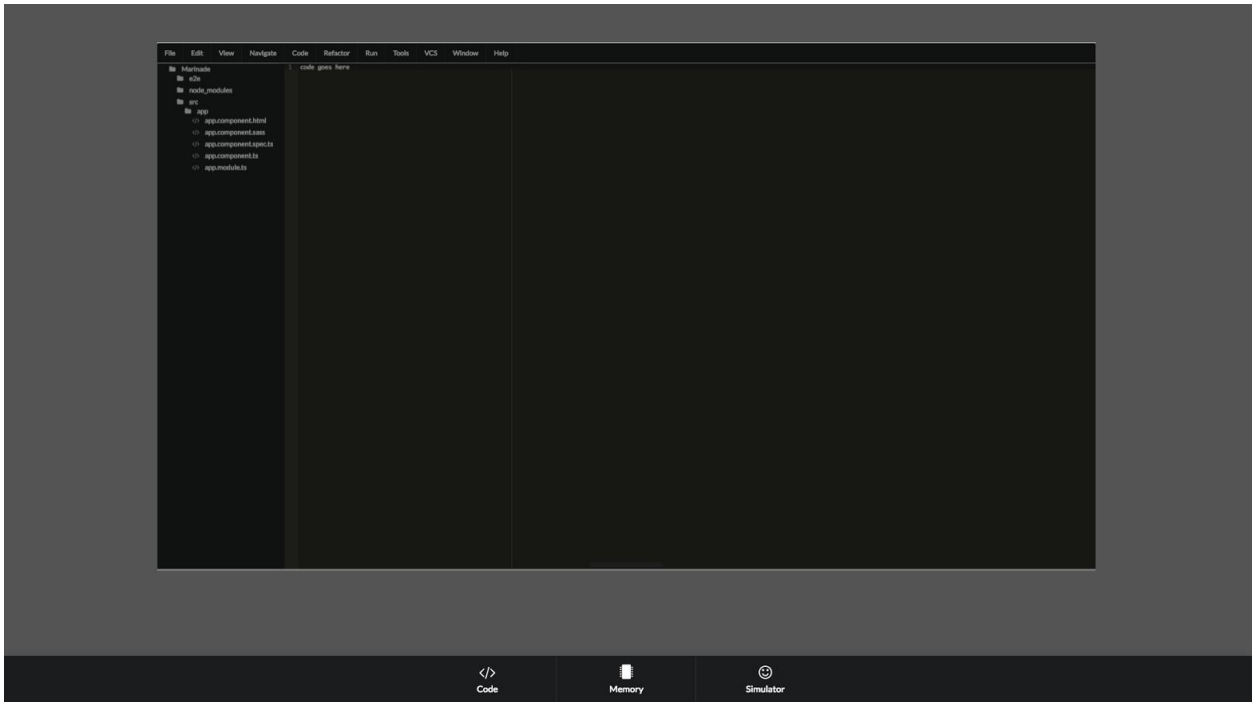

Figure 4: The task switcher view. This menu is selected using a handle at the bottom of the other three views and allows for fast switching between the code, memory, and simulator views.