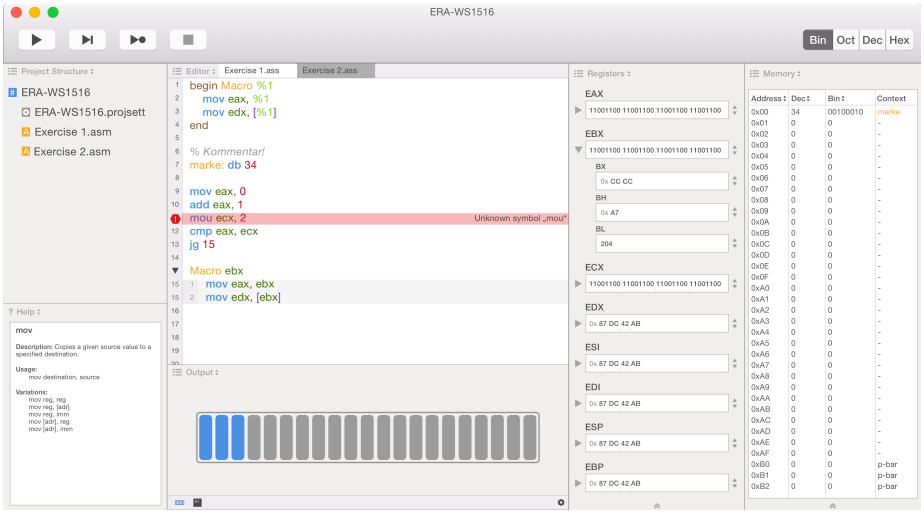

Abbildung 1: Der obige rein visuelle Entwurf soll einen groben Überblick über die geplante Umsetzung der Benutzeroberfläche geben. Im Wesentlichen ist die Aufteilung des Fensters in Toolbar und die vier Spalten mit ein bis zwei Zeilen erkennbar. Zudem wurden die einzelnen Module skizziert.

1 Graphical User Interface

Die GUI wird mit dem Qt-Framework umgesetzt, wobei grafische Elemente im Besonderen mit QML implementiert werden. Dies unterstützt die strikte Trennung von Präsentation, Steuerung und Struktur der zugrundeliegenden Daten.

Die Kommunikation mit dem *Core* erfolgt über eine C++-Schnittstelle auf Basis des Observer-Patterns.

1.1 Fenstermanagement

Die GUI soll aus verschiedenen Modulen bestehen, die zunächst als Editor-, Register-, Speicher-, Projekt-, Hilfs- und Output-Modul identifiziert wurden. Die darüberliegende Toolbar steuert Abläufe und Einstellungen, die über Modulgrenzen hinweg Relevanz haben.

Die Unterbringung der Module erfolgt in einem tabellarischen Layout bestehend aus 4 Spalten zu je 1 bis 2 Zeilen. Die untere Zeile einer Spalte kann je nach Bedarf angezeigt oder ausgeblendet werden.

Die dadurch entstehenden Modulzellen sollen variabel mit den oben beschriebenen Modulen belegt werden können, was die Möglichkeit mit einschließt, ein Modul mehrfach in verschiedenen Zellen anzuzeigen. Ein einfaches Anwendungsbeispiel für dieses Feature ist das parallele Arbeiten auf zwei Bereichen den Speichers.

1.2 Toolbar

Über die Toolbar kann die **Ausführung** des Programms gesteuert werden, wobei die Optionen *Ausführung des gesamten Programms*, *Ausführung einer einzelnen Instruktion*, *Ausführung bis zum nächsten Breakpoint* und *Abbrechen der Programmausführung* zur Verfügung stehen.

Des Weiteren lässt sich das **Zahlenformat** modulübergreifend festlegen, was insebesondere Einfluss auf Register- und Speicherinhalte hat.

1.3 Editor

Der Editor für die Eingabe der Assembler-Instruktion soll **Syntax-Highlighting** unterstützen, um Instruktions-Komponenten visuell zu trennen. Die Umsetzung erfolgt mit dem Qt-eigenen Syntax-Highlighter (*QHighlighter*), welcher mit Regex-Ausdrücken initialisiert wird. Diese werden vom Parser in Kombination mit Architektur-eigenen Keywords generiert und über den Core zur Verfügung gestellt.

Fehlermeldungen, die vom Parser ermittelt wurden, werden innerhalb der zugehörigen Zeile im Editor angezeigt, was es ermöglicht mehrere Fehlermeldungen gleichzeitig anzuzeigen.

Ein **Makroaufruf** wird im Editor als einzelne Zeile dargestellt, die bei Bedarf vom User aufgeklappt werden kann, wodurch die Makrodefinition in-place eingeblendet wird. Dies ermöglicht u.a. die schrittweise Ausführung des Codes.

1.4 Register

Die Registerwerte werden übersichtlich mit Hilfe von Byte-Separatoren dargestellt. Zu den jeweiligen Registern gehörige Unterregister können separat eingeblendet werden.

Registern wie Unterregistern kann unabhängig von den globalen Einstellungen ein eigenes Zahlenformat (binär, hexadezimal etc.) zugewiesen werden.

Intern werden die einzelnen Zahlenformate vom *Core* berechnet und als String an die GUI übergeben.

1.5 Speicher

Der Speicher teilt sich in drei Hauptbereiche: Speicheradressen, Speicherinhalte und Kontextinformationen.

Der **Adressraum** erlaubt die Unterteilung in verschiedene Datenformate, darunter Byte, Halbwort, Wort etc.

Das Zahlenformat der **Speicherinhalte** ist spaltenweise anpassbar.

Kontextinformationen zu einzelnen Speicherzellen geben Hinweise auf die Verwendung im Programm, darunter etwa Marken zu Datendefinitionen, Speicherbereiche für Ausgabegeräte und Speicher-Referenzierungen durch Register.

1.6 Projekt

Das Projekt-Modul dient der **Datei-Verwaltung**. Gegebenefalls können ein oder mehrere Dateien zu Projekten zusammengefasst werden, welche dann zusätzliche Information etwa über die verwendete Architektur sowie Speicher- und Registerinhalte speichern können.