**Proiect Arhitectura Calculatoarelor**

**SW Design document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Name | Group | Date | Changes |
| 0.1 | Circa, Dragos | 231/3 | 11.12.2018 | Initial version (basic UI) |
| 0.2 | Circa, Dragos | 231/3 |  | Done parser with data and segmentation and directives support |
| 0.22 | Circa, Dragos | 231/3 |  | Remade MPM and rest of the spreadsheet |
| 0.3 | Circa, Dragos | 231/3 |  | Remade parser from scratch. Too complex for requirements |
| 0.4 | Circa, Dragos | 231/3 |  | Drawing elements (registers, flags and memory) |
| 0.5 | Circa, Dragos | 231/3 |  | Rest of drawn elements (commands, buses, arrows) |
| 0.6 | Circa, Dragos | 231/3 |  | Added sequencer and execution flow design |
| 0.7 | Circa, Dragos | 231/3 |  | Initial sequencer implementation |
| 0.8 | Circa, Dragos | 231/3 |  | Testing |
| 1.0 | Circa, Dragos | 231/3 |  | Various fixes (seq and visual) |
| 1.1 | Circa, Dragos | 231/3 |  | Corrections in MPM and additional simulation controls |
| 1.1 | Circa, Dragos | 231/3 |  | More testing and documentation |

Contents

[Contents 3](#_Toc11256546)

[1. Scope and Purpose 4](#_Toc11256547)

[2. General Software Architecture and module interaction 5](#_Toc11256548)

[2.1. Instruction classes 5](#_Toc11256549)

[3. Function flow diagram 6](#_Toc11256550)

[3.1 Function Step() flow diagram 6](#_Toc11256551)

[4. Project specific implementations 7](#_Toc11256552)

[4.1. Architecture 7](#_Toc11256553)

[4.1.1. Instruction classes 7](#_Toc11256554)

[4.2. Module interactions 7](#_Toc11256555)

[4.3. Function flow diagrams 7](#_Toc11256556)

[4.4 Memory 8](#_Toc11256557)

[4.5 Registers 8](#_Toc11256558)

[4.6 Simulation 8](#_Toc11256559)

# Scope and Purpose

The scope of the application is to get an adequate project grade.

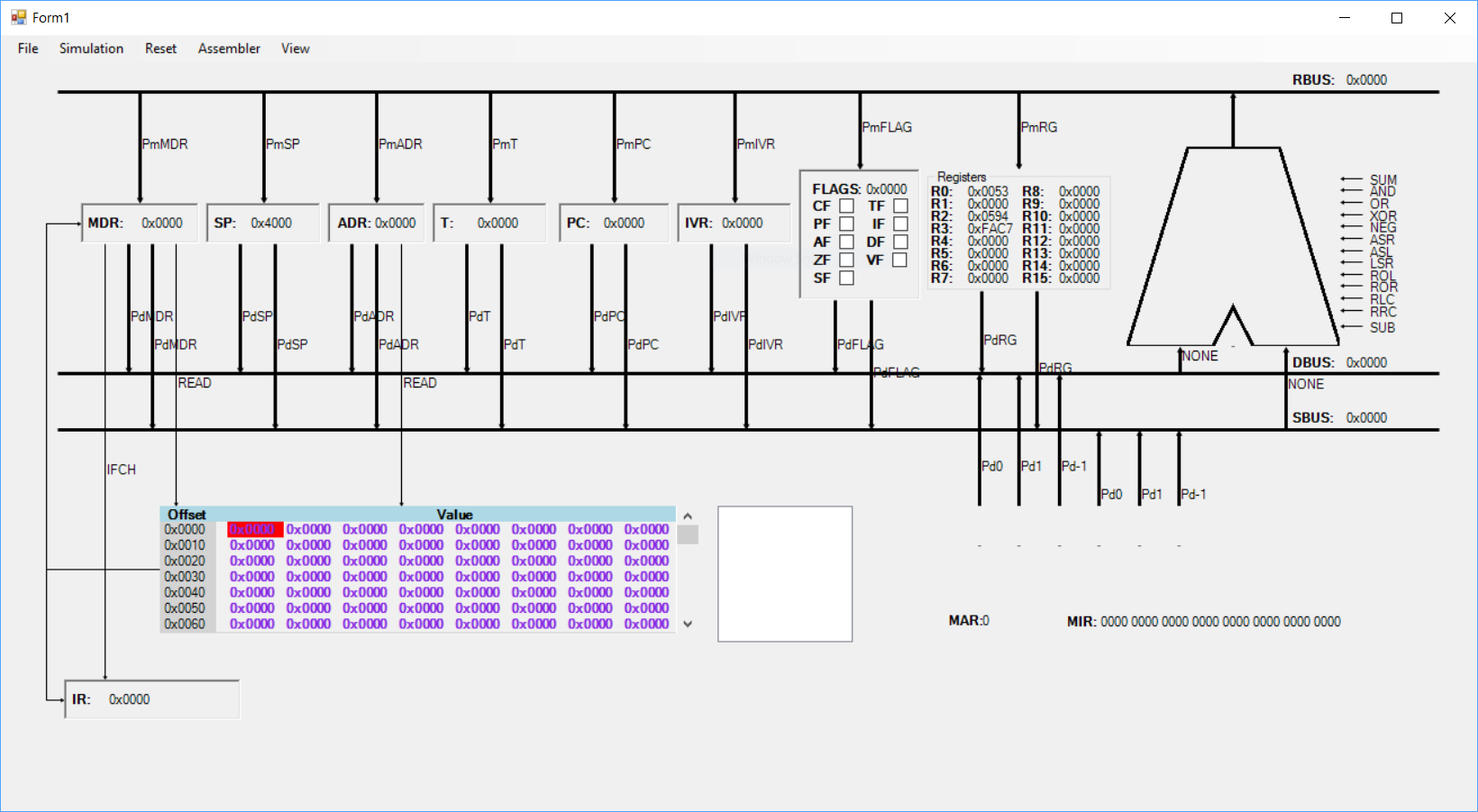


Fig 1. Application in idle state

# General Software Architecture and module interaction

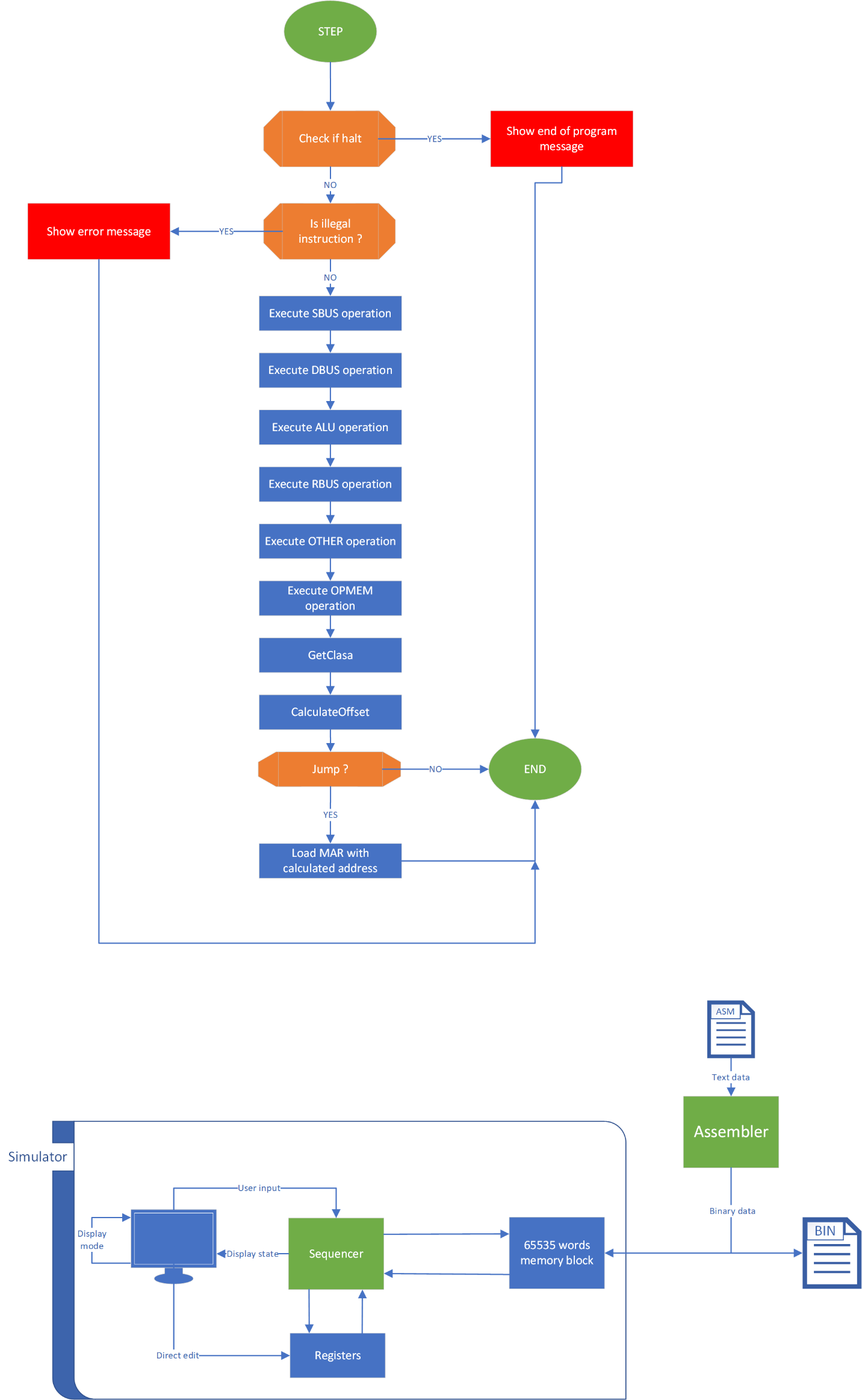


Fig 2. Module interaction. With green are marked active elements.

## Instruction classes

The classes are mapped as described in class.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instr. class | Bit 13 | Bit 14 | Bit 15 | CL1 | CL0 |
| B1 | 0 | 0 | 0 | 0 | 0 |
| B2 | 0 | 0 | 1 | 0 | 1 |
| B3 | 0 | 1 | 0 | 1 | 1 |
| B4 | 1 | 1 | 1 | 1 | 0 |

Table 1: Instruction classes

The OP codes for each instruction are available in the provided spreadsheet.

# Function flow diagram

## 3.1 Function Step() flow diagram

A screenshot of a video game

Description automatically generated

Fig 2. Step() execution flow diagram.

After each call the form is invalidated to draw the activated commands. They remain active until the next call disables them or the simulator is reset manually.

# Project specific implementations

## Architecture

The view is based around the wired version but adapted to work with the microprogrammed version.

## Instruction classes

* Class 1 (code 0): Instructions with two operands
* Class 2 (code 100): Instructions with one operand
* Class 3 (code 110): Relative jump instructions
* Class 4 (code 111): Other instructions, no operands

## Module interactions

When the user presses the “Load” button and a valid file is selected the assembler will try to assemble it. If any error occurs it will display an error message and stop parsing.

If the file is valid it will be parsed and loaded in the main memory. The user can also use the parser to just assemble a file to a standalone binary file using the option Assembler->To file. This creates a virtual memory which Is loaded with the data then dumped in the specified file.

## Function flow diagrams

There are no more interesting functions that warrant more diagrams, except the ExecuteIndex() which is spread in the last four blocks in the step diagram:

A screenshot of a video game

Description automatically generated

Fig 3. Execute index diagram

## 4.4 Memory

The memory can be accesed by the user with the MemoryView control present on the main window. Here the user can see and alter the contents of the entire memory. The navigate the user can use the scrollbar or click on the offsets to jump to an address.

## 4.5 Registers

The registers can be viewed and altered runtime such as value and display mode (binary, decimal or **hexadecimal).** The display mode can be changed from View->Registers.

The general registers view mode can not be changed.

## 4.6 Simulation

The simulation controls are as follows:

* Run: steps every 200 ms
* Stop: stops timer and resets special registers
* Pause: pauses the execution
* Step: executes Step() function

The user can also reset simulator data contained in registers , flags and memory.