



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escuela Técnica Superior de Ingeniería Informática  
Universitat Politècnica de València

# **SIMULADOR DEL COMPUTADOR EASY8**

Proyecto Final de Carrera

SIMULADOR DEL COMPUTADOR EASY8

**Autor:** David Marzo Muñoz

**Director:** Antonio Martí Campoy/Alberto González Téllez

Junio 2016

## Resumen

En este proyecto se debe realizar un simulador para ejecutar programas en ensamblador del computador Easy8.

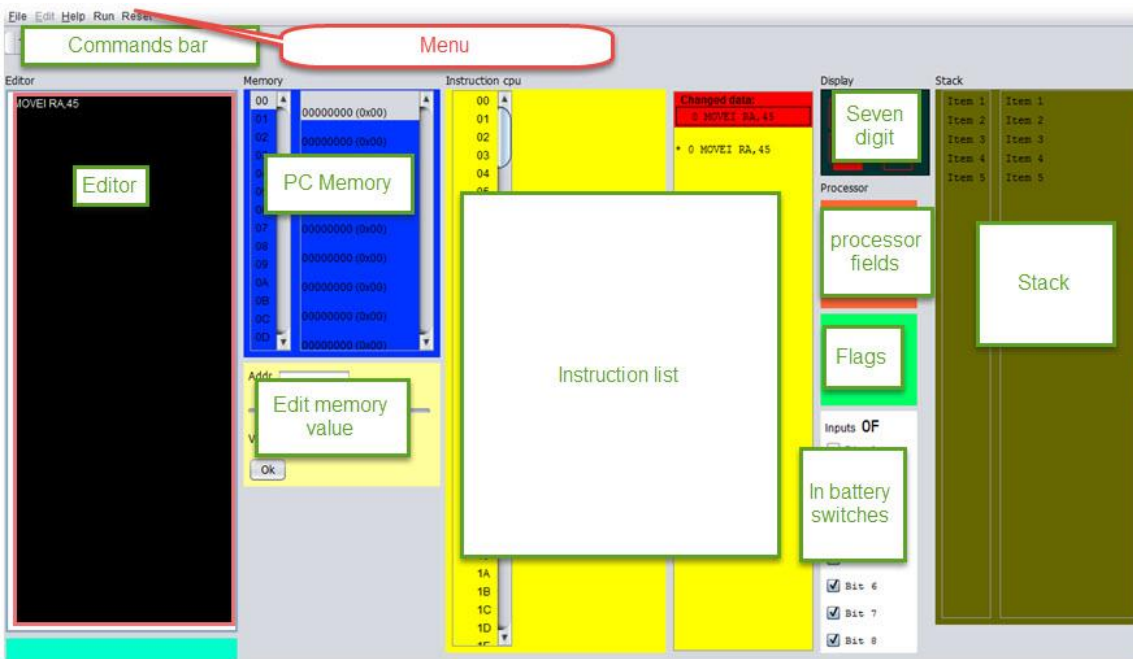
A partir de un fichero de texto plano con un programa en ensamblador, debe ensamblar y codificar el programa, para posteriormente permitir la ejecución paso a paso o completa, mostrando el estado de la memoria, de las unidades funcionales del procesador y de los periféricos incorporados en un interfaz gráfico.

**Palabras clave:** No te olvides de ponerlas ☺ integer, blandit, pharetra, urna, id.

# Tabla de contenidos

---

<b>1</b>	<b>INTRODUCTION .....</b>	<b>7</b>
<b>2</b>	<b>OBJECTIVE .....</b>	<b>8</b>
<b>3</b>	<b>REQUIREMENTS .....</b>	<b>9</b>
<b>4</b>	<b>APPLICATION DESIGN.....</b>	<b>10</b>
<b>4.1</b>	<b>Used design patterns .....</b>	<b>10</b>
4.1.1	Singleton pattern .....	10
4.1.2	Factory Pattern .....	10
<b>4.2</b>	<b>Pseudo codes .....</b>	<b>11</b>
4.2.1	Run/Step execution .....	11
4.2.2	Instruction implementation .....	11
4.2.3	Seven Digit display .....	11
4.2.4	Pseudo code Explanations .....	11
<b>4.3</b>	<b>Customization of classes .....</b>	<b>11</b>
<b>5</b>	<b>ENVIRONMENT / PROGRAMMING LANGUAGE/IMPLEMENTATION.....</b>	<b>13</b>
<b>5.1</b>	<b>Implementations .....</b>	<b>13</b>
5.1.1	Seven digit.....	13
<b>6</b>	<b>ASSEMBLE WORKING .....</b>	<b>16</b>
<b>7</b>	<b>KNOWN LIMITATIONS-IMPROVEMENTS.....</b>	<b>19</b>
	The following list contains the known limitation of the current version of the simulator.....	19
•	Error handling of asm file .....	19
•	Seven digit update .....	19
<b>8</b>	<b>MANUAL .....</b>	<b>20</b>
<b>8.1</b>	<b>Execution of Gui .....</b>	<b>20</b>
1.	Windows .....	20
2.	Linux .....	20
<b>8.2</b>	<b>Execution of assembler .....</b>	<b>20</b>
1.	Windows .....	20

2. Linux .....	20
<b>9 GUI .....</b>	<b>21</b>
In this section the GUI will be described. The gui has the following items: .....	21
<b>9.1 All parts view .....</b>	<b>21</b>
	..... 21
<b>9.2 Editing/saving assembler/memory files.....</b>	<b>22</b>
9.2.1 Open assembler file.....	22
9.2.2 Save assembler file .....	22
<b>9.3 Memory view .....</b>	<b>24</b>
9.3.1 Memory.....	24
9.3.2 Instruction cpu .....	24
9.3.3 Stack.....	24
<b>9.4 Seven segment display.....</b>	<b>25</b>
9.4.1 Battery of 8 switches .....	25
<b>9.5 Toolbar .....</b>	<b>26</b>
<b>1.1. Gui Menu .....</b>	<b>26</b>
<b>10 GUI BEHAVIOUR.....</b>	<b>27</b>
<b>10.1 File types used in the simulator.....</b>	<b>27</b>
10.1.1 Files that could be used in the Simulator .....	27
<i>The Simulator could work without any files. The files are only for storing/loading previous Assembler /memory files. ....</i>	27
<b>ASM</b> .....	27
<b>MEM</b> .....	27

10.2	How to change memory value? .....	27
11	DEBUGGER.....	29
	This section describes the options of the execution/debugger of the Simulator. ....	29
11.1	Step Button .....	29
11.2	Breakpoints.....	29
11.3	– Run Button .....	29
	Execute the instruction pointed to by the PC, update the machine status and graphic elements, and continue with the next instruction. The execution will stop when it reaches the Stop instruction or press the STOP button. ....	29
11.4	– Stop Button .....	29
12	– EASY8 INSTRUCTIONS LIST .....	30
▪	MOVEI RA, VALUE .....	30
▪	MOVR RA,25.....	30
▪	MOVE 34,RA .....	30
▪	ADDI RA,34.....	30
▪	ADD RA, 45 .....	30
▪	SUBI RA,V 56 .....	30
▪	SUB RA,46 .....	30
▪	INC RA .....	30
▪	DEC RA .....	30
▪	COMPAREI RA, VALUE .....	30
▪	COMPARE RA, VALUE .....	30
▪	JUMP ADDRESS .....	30
▪	JLESS ADDRESS .....	30
▪	JGREATER ADDRESS .....	30
▪	JEQUAL ADDRESS .....	30
▪	PUSH RA .....	30
▪	POP RA .....	30

<b>13</b>	<b>APPENDIX .....</b>	<b>31</b>
	<b>This section will give two assembler files: One with IO Out and one with IO IN. The flow of save will be displayed here also. ....</b>	<b>31</b>
<b>13.1</b>	<b>Assembler program with IO .....</b>	<b>31</b>
13.1.1	Basic flow – output to the Seven digit .....	31
13.1.2	Basic flow – Input from the seven switches battery .....	32
<b>13.2</b>	<b>Save file flow .....</b>	<b>33</b>
<b>14</b>	<b>CONCLUSIONS AND FUTURE WORK .....</b>	<b>35</b>
	<b>THIS WORK IS BASIC SIMULATOR WITH VERY SIMPLE INPUT/OUTPUT. ....</b>	<b>35</b>
	<b>INDEX .....</b>	<b>38</b>
	<b>LIST OF PICTURES .....</b>	<b>39</b>

# 1 Introduction

---

The study of the Instruction Set Architecture is a very important subject in studies of computer design and programming

First year students cannot address on real processor because of its complexity and therefore more simple computers should be used. This is the case of the Easy8 computer. Although it is an educational and very simple computer, it includes the same components like real computers: CPU with registers, main memory, input/output system and a short but representative set of instructions.

## 2 Objective

---

***The goal of this project is to make a computer simulator of the Easy8 computer as defined in FCO subject Grade Engineering Technology and Telecommunication Services ETSIT.***

**It should be easy to use by the student, with low curve learning.**

**It should be possible to be used from many platforms.**

**It should include a graphical interface.**



### 3 Requirements

---

- Execution from windows/linux/unix.
- Execution of assembler on basic assembler language with minimal set of instruction list.
- Work with ASM files, assemble them and reload them from the memory.
- Ability to stop/continue execution of the system.  
Ability to add Breakpoints.
- Ability to have input/output to display/external system.  
User interface to change memory.
- Load/save memory for working again on the same system.
- Show to the user the impact of the memory.
- Working with hexadecimal base

## 4 Application design

---

The design of the application based on object oriented while using design patterns. Frameworks were used as much as possible. Java was the selected languages, as it gave the ability to be executed from many platforms.

AGILE was used as the methodology of the writing the code.

The TDD – Test driven development was in use , partially.

The application was done with java, as object oriented language. Some design patterns were in used, as described later in internal following list. The project was done in object oriented methodology. One exemplar to this could be the Seven Digits display - which wasn't write from scratch, but used the Swing Panel and extends it.

Following framework was in used: Gui - Swing. JPanel, data model of JList were extended . XML - JAXB.

*The implementation of the handling of the instructions in the logic model – was done with TDD – Test driven development.*

***Utils class was created as a Helper class which will include the methods that are not part of the Panel handling itself. It will shortly the code to be more viewable.***

### 4.1 Used design patterns

#### 4.1.1 Singleton pattern

***The singleton patenrn was used to create only one occurrence of every member of the CPU. LogicalCpu is the class of the Singleton. Singleton implementation done with double checking to prevent entering double time on the same time.***

#### 4.1.2 Factory Pattern

***The factory pattern was used to create the required implementation of ActivityPiece in run time, for every step while execution the assembled code. Factory class name is ActivityPieceFactory.***

**1**

---

<sup>1</sup> The decision which implementation to create is done according to the input which contains the command, In the run time assumption is taken that current instruction is exist [after assemble has been pass already before]

## 4.2 Pseudo codes

### 4.2.1 Main - Run/Step execution

```
#nextAddress = getNextAddress()  
  
#step()  
  
#updatePc()  
  
#Show impact memory fields  
  
#Show next step on the Instruction CPU window
```

Explanation:

The execution method was built with the following steps:

1. Get the next instruction to be executed.
2. Execution of the step in the logic. For example: Pull data value by ref or immediately from the operand and put in the required place according to the Instruction.
3. Update the pc after the execution of the instruction.
4. Show the impact memory by the last command
5. Show the next step to be executed.

### 4.2.2 Instruction implementation

The implementation of one instruction is built with the following pseudo code:

```
#updateModel()  
#create new event for gui  
#Create jump event
```

### 4.2.3 Seven Digit display

```
# FOR INT numOfDigits <=2  
  
# FOR int pieceNumber <=7  
  
# Draw the piece ON/OFF [using the graphics.fillPolygon method]
```

### 4.2.4 Pseudo code Explanations

1. The Gui would later go on list of received events and handling them to show impact memory, next command, new registers values.
2. Create Jump event – will update the CP to another value instead of using the address of the the next command.

## 4.3 Customization of classes

1. JPanel – customized for display the Seven Digit
2. AbstractListModel – extends for adding some properties to the display of the List.



## 5 environment / programming language/Implementation

---

This section contains the selected environment/languages and implementation of one object of the Simulator.

The development was done in Windows 7. NetBeans version 8.1 was the selected IDE for this Simulator. Maven was the build system.

The selected programming language was java as it's give the ability to execute it on Windows/Linux.

The selected java version was 1.8.

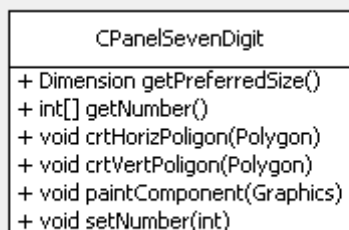
### 5.1 Implementations

#### 5.1.1 Seven digit

##### *High level design:*

The implementation of the seven digit display was done by extends the JPanel class of Swing. In the constructor of the panel we created list of polygons, every piece of the display is one polygon. Every number has list of On/OFF for the polygons. Select one value – get the required pieces that should be on and display them.

##### *UML class diagram of class*



##### *Methods*

**getPreferredSize** – used by the parent panel to create the customized panel with the required Size.

**paintComponent** - does the creation of the display.

Interface:

Creation – as in the following line, in the generated code:<sup>3</sup>

```
sdPanel = new meirdev.simulator.gui.customized.panels.CPanelSevenDigit();
```

Update value – using the method `setNumber` to update the value.

### 5.1.2 Battery of 8 switches

#### **High level design:**

Frame which contains 8 JCheckBox was created. One event handler was created for all of the array of the swing check box. The event handler was created for the **stateHandler** event. The Event handler use the method **pullInValue** for pulling the new value of the input and display it above the check boxes. When the value from the input is required – the method `pullInValue` is used.

#### **Methods**

`pullInValue` – Walk on the list of check boxes and evaluate the integer value of the array.

`extractValueFromInput` – Builds a new string which contains ZERO for off and ONE for on.

`paintComponent` -No such method. Uses default implementation of the swing components.

#### **Interface:**

Creation – Uses the designer. Every check box was created as a regular check box.

Update value – No way to update the value of the checkboxes.

Pulling value of the switches = using the method **pullInValue**.

### 5.1.3 Implementation of one instruction – we show here the instruction CALL

#### **High level design:**

The steps manager does the following activities:

- Fetches the command to be executed and set it as the current command.
- Execute the command. The execution uses the implementation of this command.
- The implementation of this command:
  - o Handle the current command – insert the pc of the next command to the stack. [Logical section of the simulator]
  - o Create a new event.

The gui does the following things:

1. Walk on list of events that are related to the Gui. No such event was created.

---

<sup>3</sup> NetBeans add the customized creation code. The component has *Custom creation code* in the Code properties tab.

2. Populate the event that was created and execute it. It will change the current pc to another one. It uses the jumpTo method of the StepManager.
- ii. The gui will change the PC to the new PC.
  - iii. The gui will update the RA with the RA register.

## 6 Config files

This section contains list of configuration issue.

### Files

- Easy8Instructions.xml – internal in the jar. This file contains the list of the available instructions of the CPU.
  - File name: Easy8Instructions.xml.
  - File location: root of the jar.

Every command has the following details in his XML entity:

1. Attribute's list of one entity:
  - a. type – the type of the instruction. There are two types – one type has additional operand, second type do not have operand value.
  - b. ov – NO for instruction with the type one\_word. Otherwise it should contains YES.
2. List of OperationList.
  - a. Instruction with some operation value would have the same name. Therefore, every operation value has one operationCode entity in the OperationList entity.
3. operationCode
  - a. Attributes:
    - i. ovValue1=type of operand.
  - b. Value – the operation code of the instruction with the specific operand.
- One change that could be done by overriding this file, without changing the code, is to replace the value of the operationCode, which contains the operation code of the assembler.

For example:

```
<operationCode>0x15</operationCode>
```

The value of 0x15 could be changed to another value,



## 7 Assemble working

---

This section will describe the abilities of the assembler, process of assembled, some of the errors..

The shell command receive input file according to the parameter of the java execution.

For example:

With the following command – it will assemble the file test.asm [the real command will be on the same line, so please do not care to the EOL which currently is inside the doc.]

```
java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.simeasy8.AssemblerReader test.asm
```

In case that file wasn't exists – it will display the error:

Parameter issue - file does not exist:[<full file name>]

So, the user should put the file according to the place which appeared in the error.

While the reading of the file:

4. No comment exists in the file.
5. Max size of the file is the memory available to this machine. Currently – the max memory which could be is 256. The reader will throw error when the number of lines will be more than 512.

In case of too many lines – the file contains more lines than 256 which is the size of instruction area – the following error will be shown:

```
Jun 20, 2016 6:25:55 PM meirdev.simulator.simeasy8.AssemblerReader verifyFileSizeSEVERE:
The file contains too much lines. :[316]. The max number could be :256
```

In case that additional space was between the command and the operand, the following error will be viewed:

```
SEVERE: Assemble Error- Operand's command contains: :[],MOVE RA,43, In line :2
```

```
Jun 20, 2016 7:00:11 PM meirdev.simulator.simeasy8.AssemblerReader assembleLines
```

The display of the error is only when the assembled was done in the shell.

In case of it will be done from the gui –

Assembled has been failed. Please fix the program lines and try again.

The error message will be disappeared only after the assemble will pass.

The Assemble stops after the first error.

## 8 Known Limitations-improvements

---

The following list contains the known limitation of the current version of the simulator.  
The current system has limitations in the following items: Error handling of Assemble process, Seven digit update. [Verification of the limit of operand value]

- Error handling of asm file

***The assemble process give good errors when the Assemble is done by the Shell/Cmd on Linux/Win systems. Please see in the manual the steps which should be done for compile asm file. When the assemble is done in the GUI – the errors will not be viewed.***

- Seven digit update

***The Seven-digit display regular definition exists only for regular decimal number. No display for the case that output value has one of the additional characters. Previous value will remain in case that RA contains value which has a characte which is part of hexadecimal base, instead of a number.***

## 9 Manual

This section describes the instruction how to execute system as gui/shell/cmd commands.

### 9.1 Execution of Gui

1. <u>Windows</u>	<ol style="list-style-type: none"> <li>1. Extract the files to working directory.</li> <li>2. Execute the Gui.bat</li> </ol>
2. <u>Linux</u>	<ol style="list-style-type: none"> <li>1. Extract the file to local folder.</li> <li>2. Open the shell to this folder</li> <li>3. Execute the following java command:</li> <li>4. <code>java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.gui.frmae.GuiSimulator</code></li> </ol>

### 9.2 Execution of assembler

1. <u>Windows</u>	<ol style="list-style-type: none"> <li>1. Extract the files to working directory.</li> <li>2. Execute the Show.bat</li> </ol>
2. <u>Linux</u>	<ol style="list-style-type: none"> <li>1. Extract the file to local folder.</li> <li>2. Open the shell to this folder</li> <li>3. Execute the following java command:</li> <li>4. <code>java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.simeasy8.AssemblerReader test.asm</code></li> </ol>

The last parameter for the java is the assembler file to be assembled.

## 10 Gui

**In this section the GUI will be described. The gui has the following items:**

Menu – Some activities that could be with the menu.

Commands bar – Main frequency activities

Editor – Editor for the Assembler program

PC Memory – Display the memory values

Edit memory value – Frame for changing value of specific memory address

Instruction menu – This frame contains List of instructions after Assemble success. Also show the next instruction to be used

Output – Seven digit - display numbers in Seven digit type of display.

Processor registers – contains the registers of the CPU.

Processor flags

Stack – show the memory of the stack.

Message []

### 10.1 All parts view

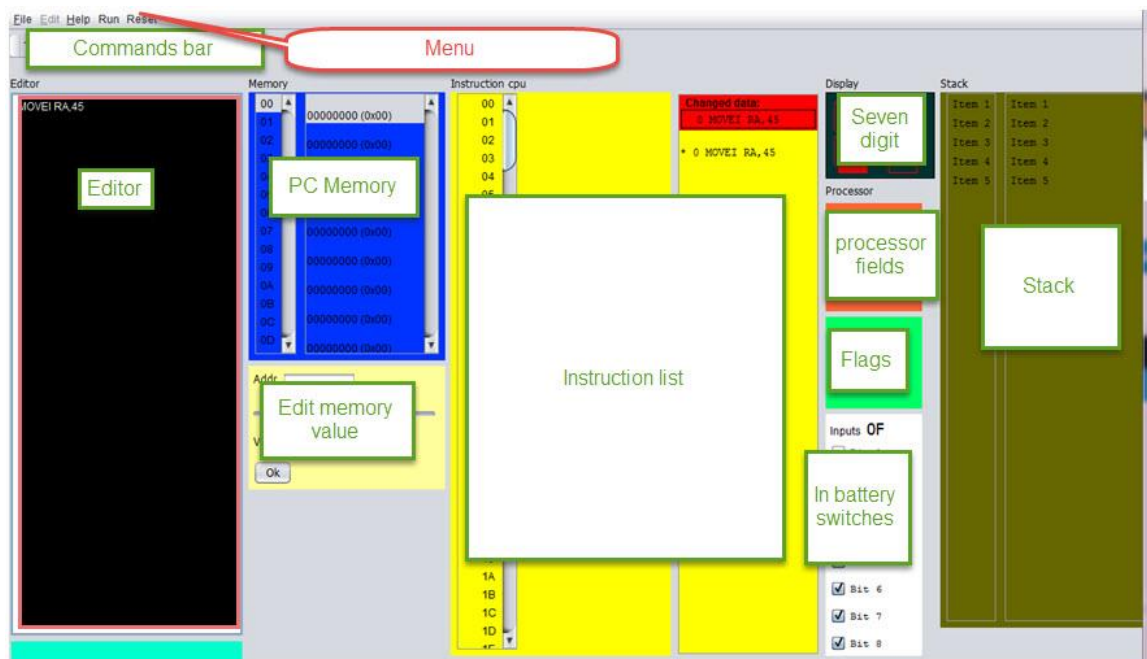


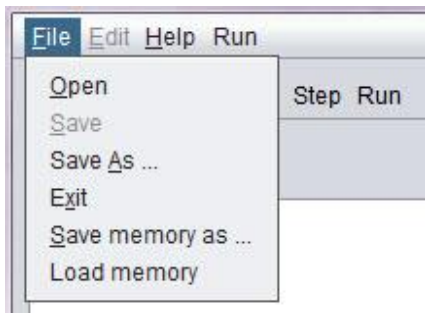
Figure 1- Simulator parts

## 10.2 Editing/saving assembler/memory files

- Edition frame to edit programs in assembly language.

### 10.2.1 Open assembler file

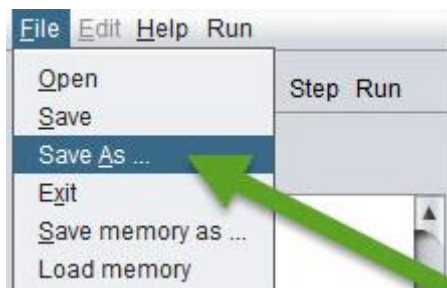
The file will be opened as in the following image, using the **File/Open** menu item /Sub menu item. Only **asm** files will be viewed from this dialog window.



2Open file

### 10.2.2 Save assembler file

Save file – Use the **Save / Save as** sub menu. Extension of the file name could be ignored in this activity.



3Save as - menu item q Sub menu item

Window (or frame) to display processor registers (PC, SP and RA) and outcome indicators (C, N, Z and V). These records will be updated either by the execution of the instructions or because the user modifies the content.

In the current version – the edit of the values is done currently in hexadecimal.

Processor

SP	0
PC	0
RA	0

Flags

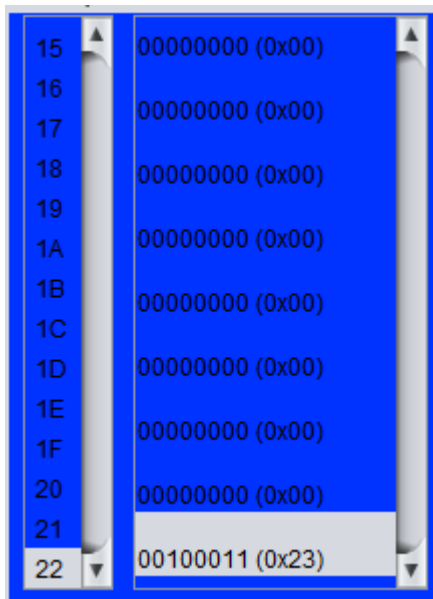
<input type="checkbox"/>	z
<input type="checkbox"/>	c

7Processor registers and outcome indicators.

## 10.3 Memory view

### 10.3.1 Memory

The memory is a frame which contains internally two lists, as in the following image:



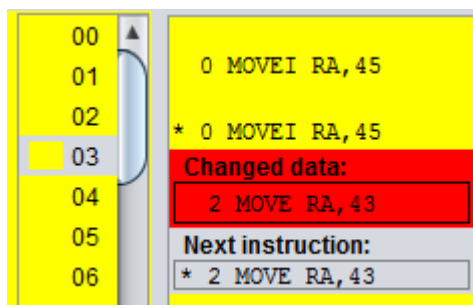
We have here two sections:

- Left list – Contains counter from ZERO to 0xFF, display as hex decimal values[in the current image – from 15 to 22 where in decimal is from 21 to 34]
- Right list – contains the memory value. The format of the memory value is [11111111] (0xFF), while 11111111 are 8 bits of the value and 0xFF contains 2digits of HEX DECIMAL value.

The two list are joined so selection of one cell in one of the lists selects also the corresponding value in the second list.

### 10.3.2 Instruction cpu

The instruction cpu is a frame which builds with two lists. The left one contains counter from ZERO to 0xFF. The right one contains the the address in two modes: when it's with '\*' – its not a starting place of an instruction.



### 10.3.3 Stack

The same frame like for the memory, but will show the stack.



## 10.4 Seven segment display

- A seven-segment display two digits to display the output of 0x01 OUT instruction (is as it is in the current simulator)

### 10.4.1 Battery of 8 switches

- A battery of 8 switches, to generate the input to the instruction IN 0x00. The user must be able to change the position of each of the switches (on / off) with the mouse.

From the high bit at the top to the lower bit .

For example:

<p><b>Inputs 4B</b></p> <p><input type="checkbox"/> Bit 1</p> <p><input checked="" type="checkbox"/> Bit 2</p> <p><input type="checkbox"/> Bit 3</p> <p><input type="checkbox"/> Bit 4</p> <p><input checked="" type="checkbox"/> Bit 5</p> <p><input type="checkbox"/> Bit 6</p> <p><input checked="" type="checkbox"/> Bit 7</p> <p><input checked="" type="checkbox"/> Bit 8</p> <p>Figure 8 battery of 8 switches - 4B value</p>	<p><b>Inputs 0F</b></p> <p><input type="checkbox"/> Bit 1</p> <p><input type="checkbox"/> Bit 2</p> <p><input type="checkbox"/> Bit 3</p> <p><input type="checkbox"/> Bit 4</p> <p><input checked="" type="checkbox"/> Bit 5</p> <p><input checked="" type="checkbox"/> Bit 6</p> <p><input checked="" type="checkbox"/> Bit 7</p> <p><input checked="" type="checkbox"/> Bit 8</p> <p>Figure 9 0F value</p>
--	--

Figure 10 battery of 8 switches - 4B value

- A hexadecimal as having the current simulator keyboard, but instead of an OK button, you will have something like a push button (<https://electrosome.com/wp-content/uploads/2012/12/Push-Button -Switch.jpg>).

Inputs **4B**

<input type="checkbox"/>	Bit 1
<input checked="" type="checkbox"/>	Bit 2
<input type="checkbox"/>	Bit 3
<input type="checkbox"/>	Bit 4
<input checked="" type="checkbox"/>	Bit 5
<input type="checkbox"/>	Bit 6
<input checked="" type="checkbox"/>	Bit 7
<input checked="" type="checkbox"/>	Bit 8

## 10.5 Toolbar

- A panel with buttons (or bar, although I prefer panel with large buttons so you can read the function of each button) with the following buttons:

Assemble, Step (Step by Step Run) Run (Uninterruptible Execution), Reset PC (Set to 0 the PC), Reset RA (Reset the RA), Reset SP (Reset the SP), Reset memory (Reset all memory) and RESET (Resets all of the above).

The RESET button in the tool bar will have confirmation, to prevent from reset by mistake.<sup>4</sup>

STOP button

### 1.1. Gui Menu

Reset menu – all the reset options.

---

<sup>4</sup> The reset menu – do not have this option. It'

# 11 Gui behaviour

---

Behavior of the different elements:

Used files in the Simulator

## 11.1 File types used in the simulator

There are two file types which are used in the simulator:

### 11.1.1 Files that could be used in the Simulator

*The Simulator could work without any files. The files are only for storing/loading previous Assembler /memory files.*

#### ASM

***This type of file contains the assembler code.***

1. Could be loaded from param when execution is from console or by Menu item when execution is done by the Gui.

The format of the file is ascii, separated by colon

#### MEM

***This type of file contains the memory code.***

The format of the file is ascii, separated by colon

-buttons Reset PC, RA, SP and memory.

These buttons reset the corresponding record (or all memory). The outcome indicators are reset in conjunction with RA or when reset everything.

Assembler –button

Once introduced into an assembly program editor, the simulator must analyze it to translate into machine code. If an error is detected, for example, the instruction does not exist or is missing an operand, display an error message and indicate where the error occurred. It takes no great sophistication, as the language is simple and regular.

If no errors are loaded into memory the result of code the program into machine code.

## 11.2 How to change memory value?

Use one of the following ways to change the value of the memory:

- a. Click on the selected memory value

- i. One click – change the selected memory value, and change also the selected address in the Yellow cube
  - ii. Double click – Move the focus of the frame to the input holding the new value of the memory, after the selected address already changed to the correct address.
- b. Change the slider “val” of the memory value.

The value which appears on the input is the decimal value, on the memory itself – it's appeared in hex decimal value.

## 12 Debugger

---

This section describes the options of the execution/debugger of the Simulator.

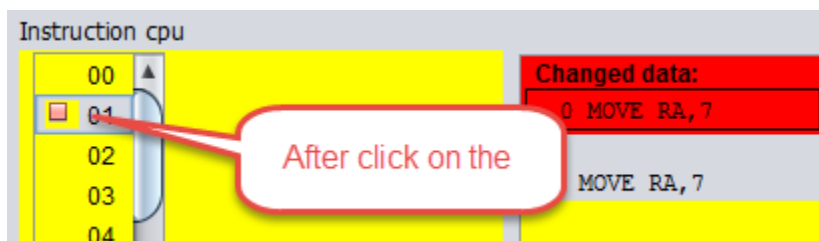
### 12.1 Step Button

Execute the instruction pointed to by the PC and stop updating all the graphics and state of the computer elements.

### 12.2 Breakpoints

The user could add a break point in the required address.

*NOTE:* Breakpoint could be added only after finished of assemble activity.



### 12.3 – Run Button

Execute the instruction pointed to by the PC, update the machine status and graphic elements, and continue with the next instruction. The execution will stop when it reaches the Stop instruction or press the STOP button.

### 12.4 – Stop Button

Stops program execution.

## 13 – Easy8 instructions list

---

This section show the list of instructions exist in the Easy8.

- When command ends with the char: 'T' it means that it will populate the value in the second operator and use it to the specific instruction.
  - MOVEI RA, VALUE
  - MOVR RA,25
  - MOVE 34,RA
  - ADDI RA,34
  - ADD RA, 45
  - SUBI RA,V 56
  - SUB RA,46
  - INC RA
  - DEC RA
  - COMPAREI RA, VALUE
  - COMPARE RA, VALUE
  - JUMP ADDRESS
  - JLESS ADDRESS
  - JGREATER ADDRESS
  - **JEQUAL** ADDRESS
  - PUSH RA
  - POP RA
  - CALL ADDRES
  - RET

14

---

14

## 14 Appendix

This section will give two assembler files: One with IO Out and one with IO IN. The flow of save will be displayed here also.

### 14.1 Assembler program with IO

#### 14.1.1 Basic flow – output to the Seven digit

This flow will display Hex 77 in the Seven Digit output.

The following steps should be done to work with ASM file.

The display of the seven digit for the default value will be as in the following image:

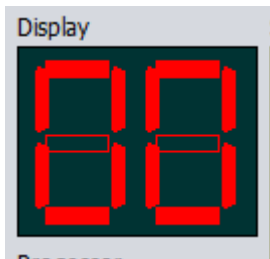


Figure 11Seven digit contain default value

1. Load ASM file with Open file menu/Sub menu
2. Edit the file in the Editor section
3. Clear the contents of the editor
4. Add the following lines:
  - a. OUT 77
  - b. STOP

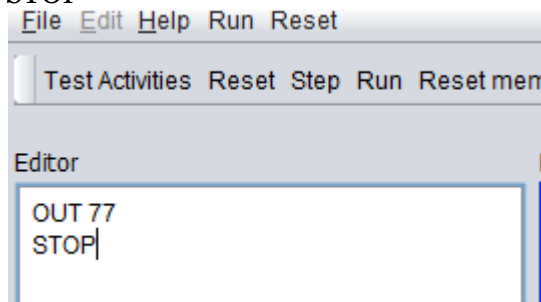


Figure 12Editie -ASM lines to display 77 in the seven digit

5. Assemble the file.
6. Run the assembled code.

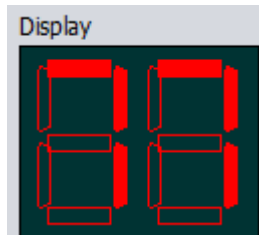


Figure 13 Seven digit in the ending of the execution - hex 77

#### 14.1.2 Basic flow – Input from the seven switches battery

The following steps should be done to work with ASM file.

The Battery switch list should be as in the following image:

(Only value that has only regular digits will be displayed in the Seven Digit display)

##### Inputs 24

- ☐ Bit 1
- ☐ Bit 2
- ☒ Bit 3
- ☐ Bit 4
- ☐ Bit 5
- ☒ Bit 6
- ☐ Bit 7
- ☐ Bit 8

1. Clear the contents of the editor
2. Add the following lines:
  - a. IN 5
  - b. OUT 77
  - c. STOP
3. Assemble the file.
4. Run the assembled code.

The following image, contains the registers as they are appeared in the end of the program:



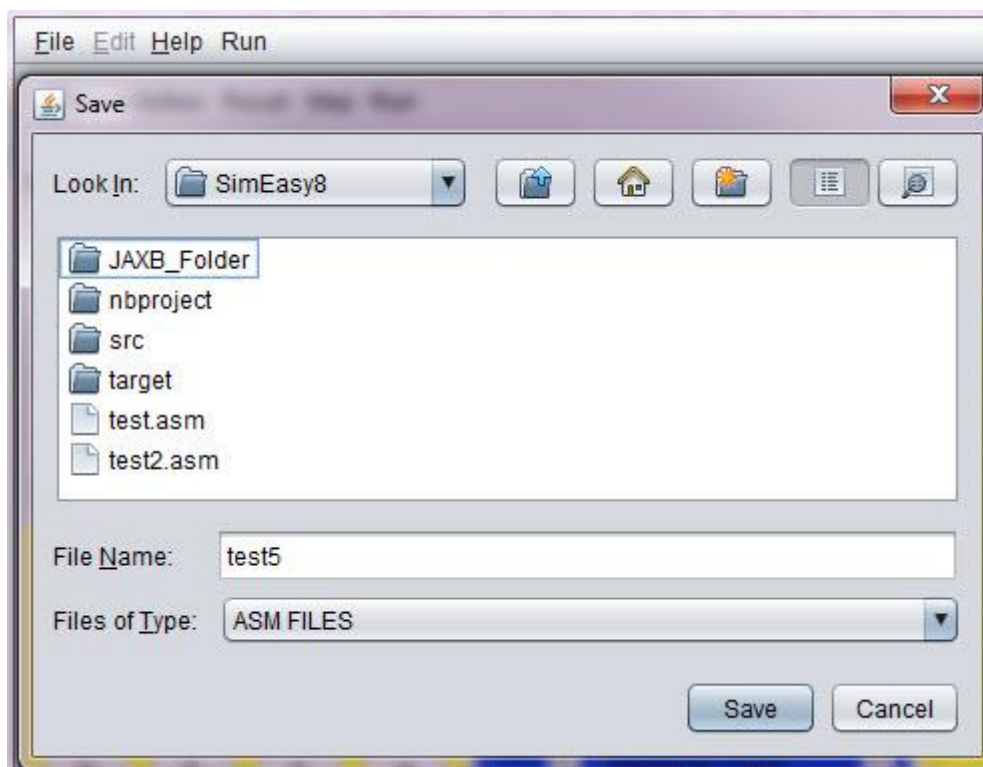
The instruction list will be as in the following image



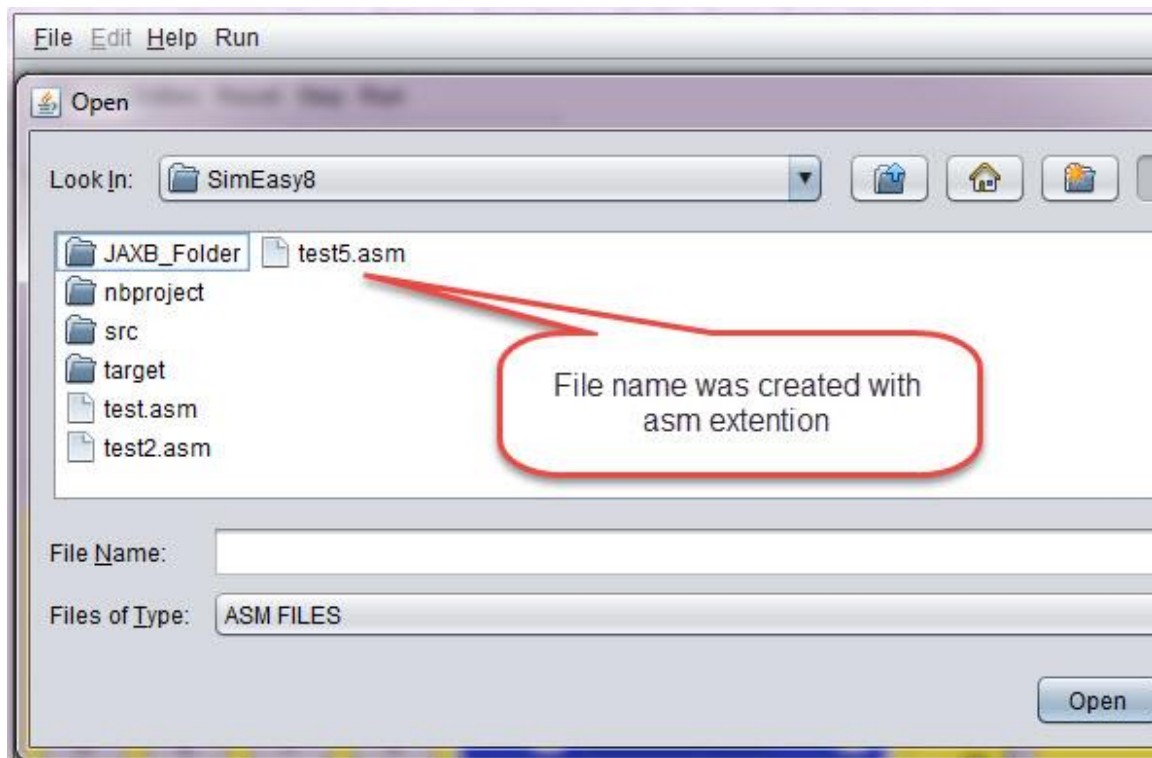
```
0 IN 5
* 0 IN 5
2 OUT 77
* 2 OUT 77
Changed data:
4 STOP
```

## 14.2 Save file flow

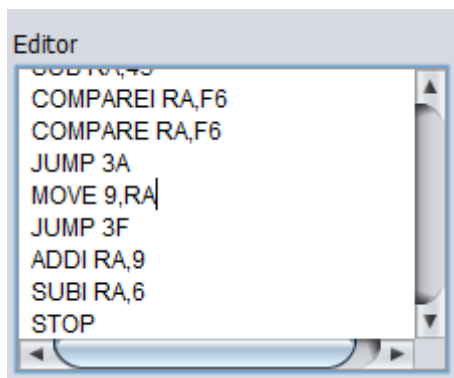
Saving the file name with no extension, will save the file with the correct extension, as in the following pictures. The result is – that open file will view the file and file will be opened with no issues:



4save asm file - test5 - with no extension



5The file was saved correctly - as test5.asm[Viewdlated by the open file ]



6Editor window

The editor window – contains window with editor for the ASM code.

## 15 CONCLUSIONS and future work

---

This work is basic simulator with very simple input/output.

Additional improvements - already exist in another section.

New features could be storing the asm file in the internet, grouped by classes.

Programing improvements – could be do the next step and split the code of the logical model and the GUI itself, so another model could be used.

### **Comment**

**For adding a new command – one existing command should be removed.**





## Index

Execution: Asseble of asm file, 15

Seven Digit: comment, 27

## List of pictures

<a href="#">Figure 1Seven digit contain default value - 00</a> .....	30
<a href="#">Figure 2Editie -ASM lines to display 77 in the seven digit</a> .....	30
<a href="#">Figure 3Seven digit in the ending of the execution - hex 77</a> .....	30
<a href="#">Figure 4Seven digit contain default value - 00</a> .....	31
<a href="#">Figure 5Seven digit in the ending of the execution - hex 77</a> .....	31
<a href="#">Figure 6- Simulator parts</a> .....	18
<a href="#">7Open file</a> .....	19
<a href="#">8Save as - menu item q Sub menu item</a> .....	19
<a href="#">9save asm file - test5 - with no extension</a> .....	20
<a href="#">10The file was saved corectly - as test5.asm[Viewdlated by the open file ]</a> .....	20
<a href="#">11Editor window</a> .....	21
<a href="#">12Processor registers and outcome indicators</a> .....	21
<a href="#">Figure 13 battery of 8 switches - 4B value</a> .....	24
<a href="#">Figure 14 oF value</a> .....	24
<a href="#">Figure 15 battery of 8 switches - 4B value</a> .....	24

/\*

1. Introduction 2. Objective 3. Requirements 4. Design of the application 5. Implementation 6. User Manual 7. Conclusions 8. Bibliography I mention what I would each of these sections. 1. Introduction: where you say that the study of the Instruction Set Architecture is very important in studies .... but first course can not address a real processor because of its complexity and therefore ... use simplifications, educational systems that do not really exist ... .. and to do simulators. 2. Objective: To say that the goal is to make a computer simulator Easy8 as defined in FCO subject Grade Engineering Technology and Telecommunication Services ETSIT. It is also easy to use by the student and if possible platform. And for that you propose the development of a Java application with graphical interface. 3. Requirement: You put a list of what you want to do the application, in all its facets: edit, assemble, execute step by step graphical interface, input and output ... 4. Application design: more or less what has commanded me, but divided into sections. An introduction with an image of the different elements of the interface, and then a section for each of them: editor, view memory, I / O, etc .. 5. Implementation: explains the language and tools used for implementation. If you have made modular programming you put the different files that have organized the source code. And usually put a list of the functions that you created. Not put the code, only the prototype of

the functions and a short description of the function. This list of functions you leave for last. 6. Manual: for that, a short manual with steps to use the simulator. 7. Conclusions 8. Bibliography

\*/