Table of Contents

Introduction

The study of the Instruction Set Architecture is very important in studies of computer design and prrograming.
First course cannot address on real processor because of its complexity and therefore the Easy8 processor is been used. Main items, like input, assign/get values from the registers, memory exist in this simple CPU.

Objective

The goal of the system 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 to work with.
In this system we built is with java, so it could be used in window/linux/unix. The development of a Java application will be with graphical interface.

Requirements

1. Execution of compiler on basic assembler with minimal set of instruction list.
2. Work with ASM files, compile them and reload them from the memory.
3. Ability to stop/continue execution of the system.
4. Ability to add Breakpoints.
5. Ability to have input/output to display/external system.
6. User interface to change memory.
7. Load/save memory for working again on the same system.
8. Show to the user the impact of the memory
9. Working with hex decimal base

Application design

The application was done with java, using some design patterns. The project was done in object oriented methodology. Following framework was in used: Gui – Swing. JPanel, data model of JList were extended . XML - JAXB.

<u>Used design patterns:</u>

1. Singleton pattern – Mainly for handling just one occurrence of every member of the CPU. LogicalCpu is the class of the Singleton. Sigleton implementation done with double checking to prevent entering double time on the same time.
2. Factory Pattern – to create the implementation of ActivityPiece in run time, for every step while execution the assembled code[1]. Factory class name is ActivityPieceFactory.

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

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

Following pseudo code contains the execution of the Step while running [or run till breakpoint or run just one step]

```
#nextAddress = getNextAddress()
#Execution step()
#updatePc() → the pc after the execution of the command
#Show impact memory fields
#Show next step on the Instruction CPU window
```

The implementation of command is build from three parts:

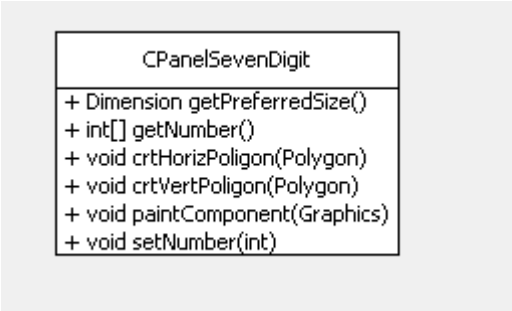
1. Update the model internally
2. Create event that the Gui should handle [to update some views of the Simulator]
3. Create Jump event – will update the CP to another value instead of the next command.

The Gui – goes on all the GuiEvents and handle all of them as well
The Gui – verify if exist Jump event and if so – do the required changes regarding to the PC.

Extensions to swing classes

1. Seven digit implementation:

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.



getPreferredSize – used by the parent of the Seven digit panel to create the panel with the required Size.

paintComponent Pseudo code– does the creation of the display.

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

Interface:

Creation – as in the following line, in the generated code:[2]

[2] NetBeans add the customized creation code. The component has ***Custom creation code*** in the Code properties tab.

sdPanel = new meirdev.simulator.gui.customized.panels.CPanelSevenDigit();
Update value – using the method setNumber to update the value.

Known Limitations-improvements that could be done

1. The compile process with good error viewing is viewable only in the console. Please see in the manual the steps which should be done for compile asm file.
2. 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 one character, instead of a number.

Manual

Execution of Gui

Windows	1. Extract the files to working directory. 2. Execute the Gui.bat
Linux	Extract the file to local folder. Open the shell to this folder Execute the following java command: **java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.gui.frmae.GuiSimulator**

Execution of compiler

Windows	1. Extract the files to working directory. 2. Execute the Show.bat
Linux	Extract the file to local folder. Open the shell to this folder Execute the following java command: **java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.simeasy8.AssemblerReader test.asm**

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

Basic flow – output to the Seven digit

Basic flow – execution of a program that will display hex value of 77.
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:

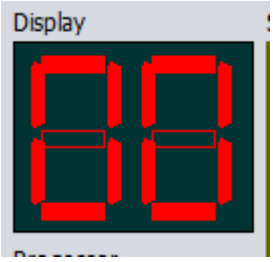


Figure 1Seven digit contain default value - 00

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

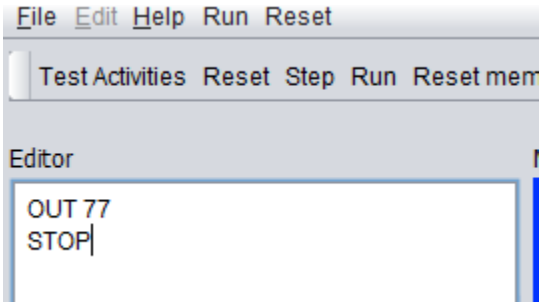


Figure 2Editie -ASM lines to display 77 in the seven digit

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

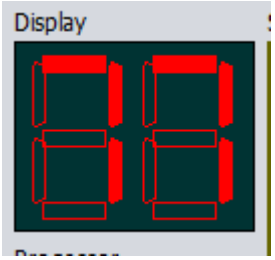


Figure 3Seven digit in the ending of the execution - hex 77

Basic flow – Input from the seven switches battery

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:

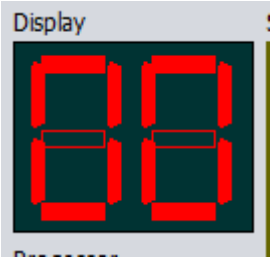


Figure 4Seven digit contain default value - 00

7. Load ASM file with Open file menu/Sub menu
8. Edit the file in the Editor section
9. Clear the contents of the editor
10. Add the following lines:
    a. IN
    b. OUT RA
    c. STOP

11. Assemble the file.
12. Run the assembled code.

Figure 5Seven digit in the ending of the execution - hex 77

Gui

Parts view

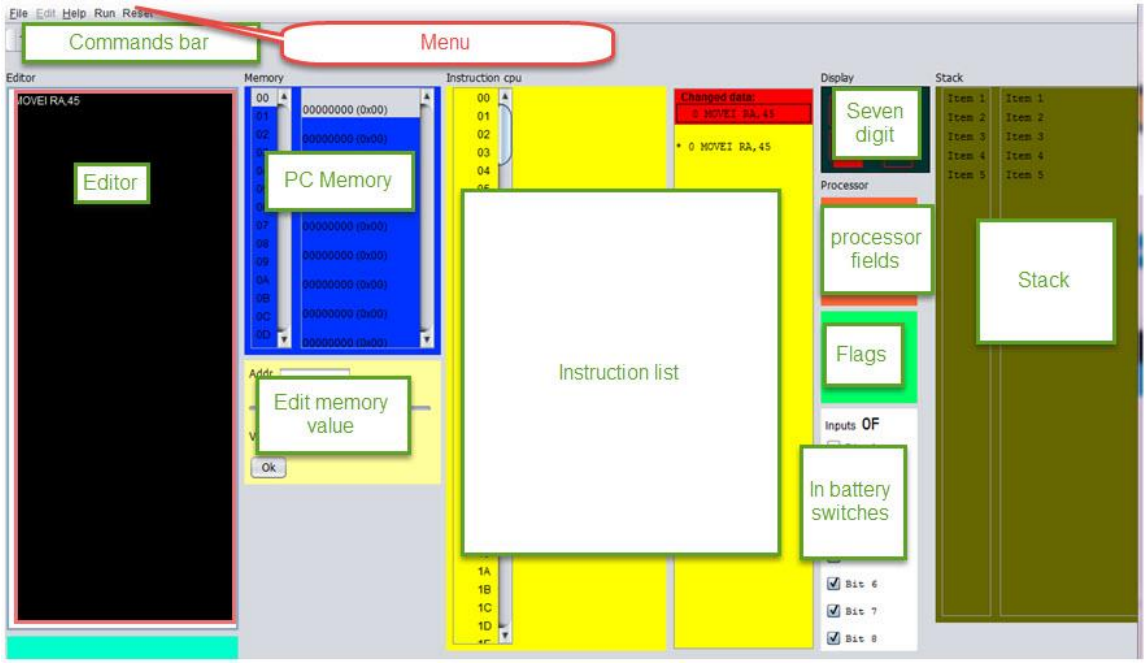


Figure 6- Simulator parts

File types used in the simulator

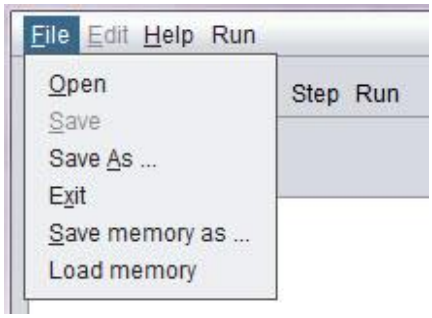
There are two file types which are used in the simulator:
ASM – This type of file contains the assembler code.
MEM – this type of file contains the memory code.
The graphic interface must include the following elements:

Editing/saving assembler

- Window (or frame) editing to introduce programs in assembly language.
Buttons to save disk drive or loaded from an assembly program.

Open assembler file

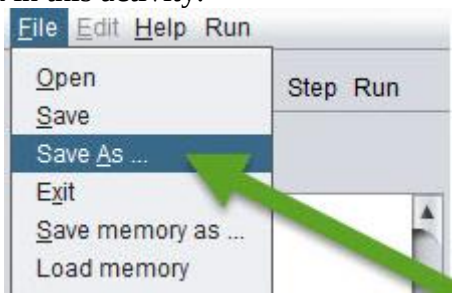
The file will be opened as in the following image, using the ***File/Open*** menu item /Sub menu item. Only ***asm*** files will be viewed from this dialog window.



7Open file

Save assembler file

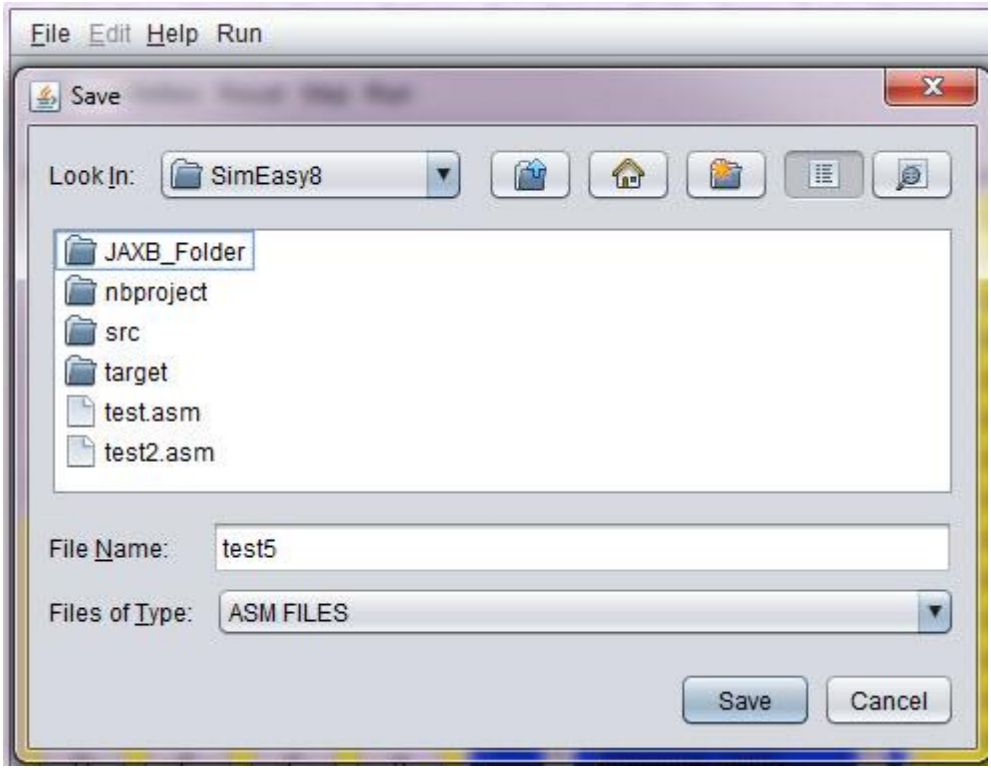
2. Save file – Use the ***Save*** / ***Save as*** sub menu. Extention of the file name could be ignored in this activity.



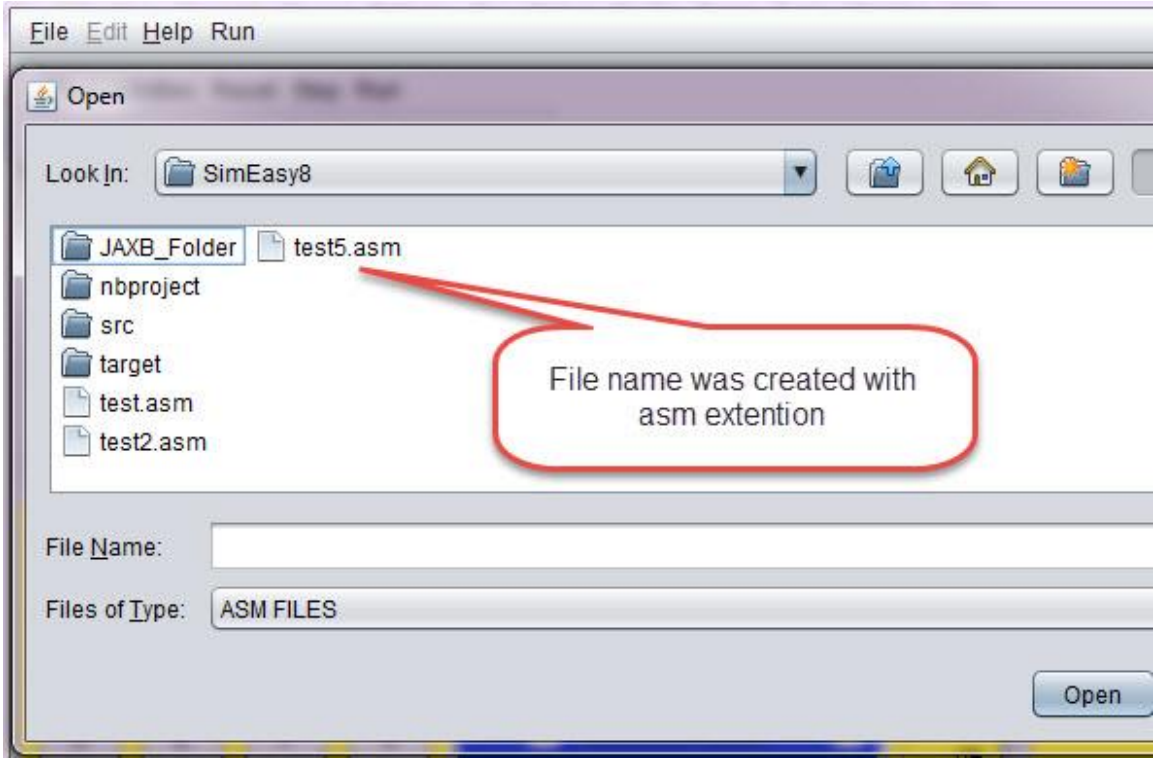
8Save as - menu item q Sub menu item

Save file flow:

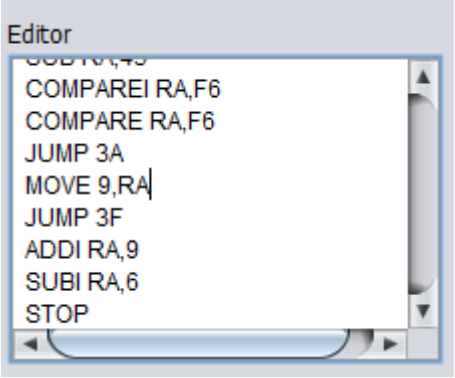
Saving the file name with no extention, will save the file with the corect extention, as in the folowing pictures. The result is – that open file will view the file and file will be opened with no issues:



9save asm file - test5 - with no extension



10The file was saved corectly - as test5.asm[Viewdlated by the open file]

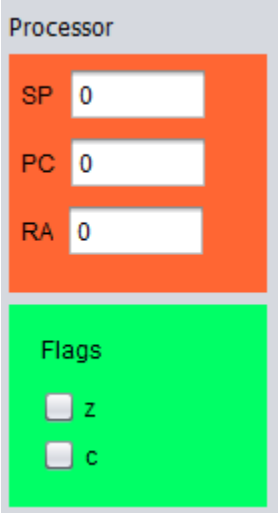


11Editor window

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

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. You should view / edit both binary and hexadecimal.

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



12Processor registers and outcome indicators.

Memory view

Memory

Instruction cpu

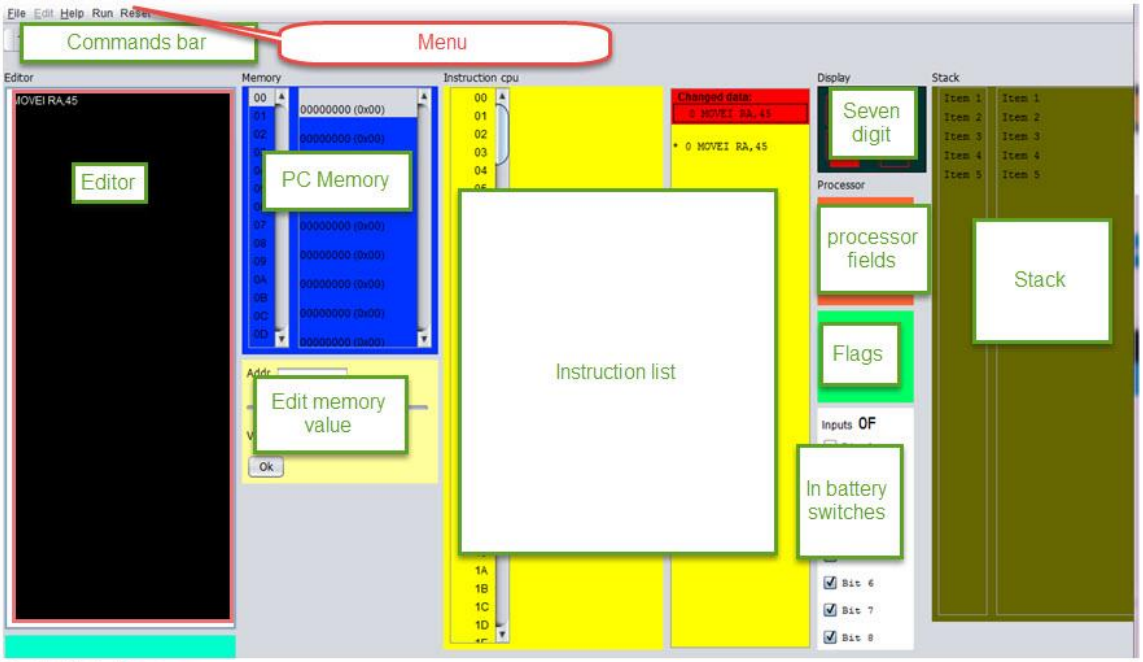
Stack

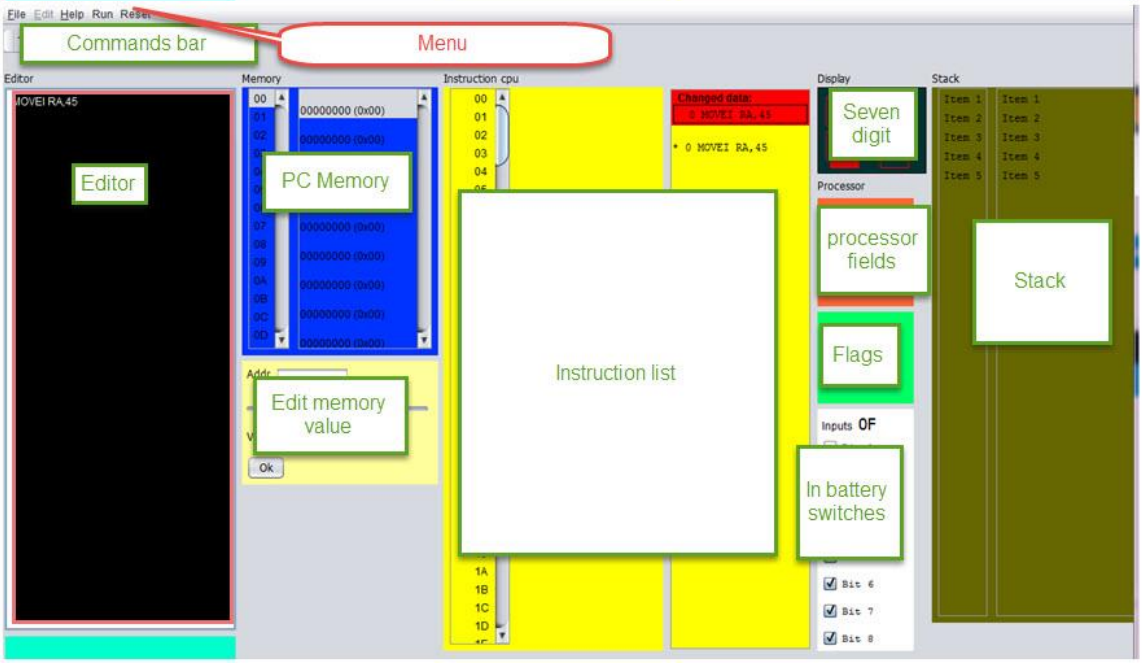
- Two windows (or frames) to display the computer's memory (256 bytes). This allows the student displayed at the same time the bottom of memory where your program into machine code, and in the other window another address range, which may have data or be the stack. Both memory addresses and their contents are displayed in hexadecimal. It must be possible to modify the contents of a memory address. The way it is now does not quite like, but not if it is possible to make a memory location by clicking on its value and changing it is changed.

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)

- A peripheral output to your liking. Something chart, which will send a byte or two and change color, shape or position. For example, a needle that moves right or left depending on the value that you send it ... – Only thing wasn't implemented





Input 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:

Figure 13 battery of 8 switches - 4B value

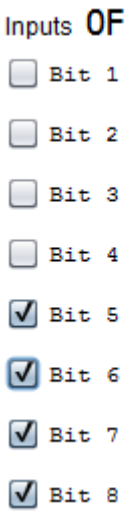
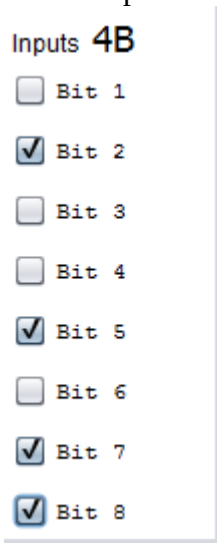


Figure 14 0F value

Figure 15 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).



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.[i]
STOP button

Gui Menu

Reset menu – all the reset options.

Gui behaviour

Behavior of the different elements:

-buttons Reset PC, RA, SP and memory.
These buttons should reset the corresponding record (or all memory). The outcome indicators are reset in conjunction with RA or when reset everything.

Joining -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.

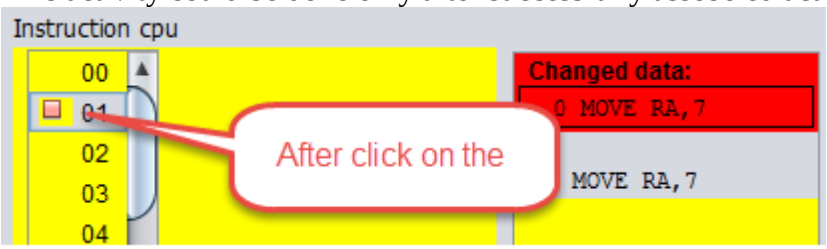
Debugger

- Step Button

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

- Breakpoint [Wasn't part of the requirement]

The user could add break point in the required address.
This activity could be done only after successfully assebeled activity.



- 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 button or press the STOP instruction.

- Stop Button

Stops program execution.

- Records and memory must change according to the execution of instructions (and if the user forces a value, of course)

- It would be very interesting that when a memory location is changed in the second window that shows the memory will be displayed, if it is not already, the memory area around the modified position.

- When a step execution is done, it should light or at least an arrow pointing instruction to be executed. If you can do both in the assembly code and source code would be fine.

Instructions to run:

In addition to the instructions in the documentation, including byte (which is not an instruction, is a thing of assembler) are two more who would want will implement instructions. And CLEAR are COMPAREP PORT PORT, but these'll talk later, when you start scheduling the execution of instructions.

The code to update indicators C, N, Z and V you'll pass, because although it is not complex is not trivial.

TABLE 1

EASY8 INSTRUCTION SET.

Instruction	Operation code	Description
MOVEI RA, VALUE		
MOVE RA, [ADDRESS]		
MOVE [ADDRESS], RA		
ADDI RA, VALUE		
ADD RA, [ADDRESS]		
SUBI RA,VALUE		

SUB RA, [ADDRESS]		
INC RA		
DEC RA		
COMPAREI RA, VALUE		
COMPARE RA, [ADDRESS]		
JUMP ADDRESS		
JLESS ADDRESS		
JGREATER ADDRESS		
JEQUAL ADDRESS		
PUSH RA		
POP RA		
CALL ADDRES		
RET		

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.

Index

/*

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

*/

---

[i] The reset menu – do not have this option. It’