**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

[1 Introduction 7](#_Toc453786404)

[2 Objective 8](#_Toc453786405)

[3 Requirements 9](#_Toc453786406)

[4 Application design 10](#_Toc453786407)

[4.1 Used design patterns 10](#_Toc453786408)

[4.1.1 Singleton pattern 10](#_Toc453786410)

[4.1.2 Factory Pattern 10](#_Toc453786411)

[4.2 Pseudo codes 11](#_Toc453786412)

[4.2.1 Run/Step execution 11](#_Toc453786413)

[4.2.2 Instruction implementation 11](#_Toc453786414)

[4.2.3 Seven Digit display 11](#_Toc453786415)

[4.2.4 Pseudo code Explanations 11](#_Toc453786416)

[4.3 Customization of classes 11](#_Toc453786417)

[5 environment / programming language/Implementation 12](#_Toc453786430)

[5.1 Implementations 12](#_Toc453786431)

[5.1.1 Seven digit 12](#_Toc453786432)

[6 Known Limitations-improvements 14](#_Toc453786433)

[The following list contains the known limitation of the current version of the simulator. 14](#_Toc453786434)

[ Error handling of asm file 14](#_Toc453786435)

[ Seven digit update 14](#_Toc453786436)

[7 Manual 15](#_Toc453786437)

[7.1 Execution of Gui 15](#_Toc453786438)

[1. Windows 15](#_Toc453786439)

[2. Linux 15](#_Toc453786440)

[7.2 Execution of assembler 15](#_Toc453786442)

[1. Windows 15](#_Toc453786443)

[2. Linux 15](#_Toc453786444)

[8 Gui 16](#_Toc453786445)

[In this section the Gui will be described. The ways to do every option. 16](#_Toc453786446)

[8.1 All parts view 16](#_Toc453786447)

[ 16](#_Toc453786448)

[8.2 File types used in the simulator 16](#_Toc453786449)

[8.2.1 Files that could be used in the Simulator 16](#_Toc453786450)

*[The Simulator could work without any files. The files are only for storing/loading previous Assembler /memory files.](#_Toc453786451)* [16](#_Toc453786451)

**[ASM](#_Toc453786452)** [16](#_Toc453786452)

**[MEM](#_Toc453786453)** [16](#_Toc453786453)

[8.3 Editing/saving assembler/memory files 17](#_Toc453786455)

[8.3.1 Open assembler file 17](#_Toc453786457)

[8.3.2 Save assembler file 17](#_Toc453786458)

[8.4 Memory view 19](#_Toc453786459)

[8.4.1 Memory 19](#_Toc453786460)

[8.4.2 Instruction cpu 19](#_Toc453786461)

[8.4.3 Stack 19](#_Toc453786462)

[8.5 Seven segment display 19](#_Toc453786463)

[8.5.1 Battery of 8 switches 20](#_Toc453786464)

[8.6 Toolbar 21](#_Toc453786465)

[1.1. Gui Menu 22](#_Toc453786466)

[9 Gui behaviour 23](#_Toc453786467)

[9.1 How to change memory value? 23](#_Toc453786468)

[10 Debugger 24](#_Toc453786469)

[This section describes the options of th exection/debugger of the Simulator. 24](#_Toc453786470)

[10.1 Step Button 24](#_Toc453786471)

[10.2 Breakpoints 24](#_Toc453786472)

[10.3 – Run Button 24](#_Toc453786473)

[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. 24](#_Toc453786474)

[10.4 – Stop Button 24](#_Toc453786475)

[11 – Easy8 instructions list 25](#_Toc453786476)

[ MOVEI RA, VALUE 25](#_Toc453786477)

[ MOVR RA,25 25](#_Toc453786478)

[ MOVE 34,RA 25](#_Toc453786479)

[ ADDI RA,34 25](#_Toc453786480)

[ ADD RA, 45 25](#_Toc453786481)

[ SUBI RA,V 56 25](#_Toc453786482)

[ SUB RA,46 25](#_Toc453786483)

[ INC RA 25](#_Toc453786484)

[ DEC RA 25](#_Toc453786485)

[ COMPAREI RA, VALUE 25](#_Toc453786486)

[ COMPARE RA, VALUE 25](#_Toc453786487)

[ JUMP ADDRESS 25](#_Toc453786488)

[ JLESS ADDRESS 25](#_Toc453786489)

[ JGREATER ADDRESS 25](#_Toc453786490)

[ JEQUAL ADDRESS 25](#_Toc453786491)

[ PUSH RA 25](#_Toc453786492)

[ POP RA 25](#_Toc453786493)

[12 Appendix 26](#_Toc453786538)

[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. 26](#_Toc453786539)

[12.1 Assembler program with IO 26](#_Toc453786540)

[12.1.1 Basic flow – output to the Seven digit 26](#_Toc453786542)

[12.1.2 Basic flow – Input from the seven switches battery 27](#_Toc453786543)

[12.2 Save file flow 28](#_Toc453786545)

[13 CONCLUSIONS and future work 30](#_Toc453786546)

[This work is basic simulator with very simple input/output. 30](#_Toc453786547)

[Index 33](#_Toc453786548)

[List of pictures 34](#_Toc453786549)

# Introduction

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

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.

# 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.

# 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**
* **Show to the user the impact of the memory.**
* **Edit with hex decimal values**ues**.**

# Application design

The design of the application based on object oriented while using design patterns. Frameworks were used as much as possible.

AGILE was used as the methodology of writing the code.

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.So instead of creation of new framework that will include Seven Digits display – extend of the Panel of the Swing framework was done.

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.***

## Used design patterns

### Singleton pattern

Will be used for handling just 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.

### Factory Pattern

to create the implementation of ActivityPiece in run time, for every step while execution the assembled code. Factory class name is ActivityPieceFactory.  
[[1]](#footnote-2)

## Pseudo codes

### 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.
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.

### Instruction implementation

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

#updateModel()

#create new event for gui

#Create jump event

### Seven Digit display

# FOR INT numOfDigits <=2

# FOR int pieceNumber <=7

# Draw the piece ON/OFF [using the graphics.fillPolygon method]

### 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.

## Customization of classes

1. JPanel – customized for display the Seven Digit

# 

# 

# 

# 

# 

# 

# 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 programing language was java as it’s give the ability to execute it on Windows/Linux.

Java version was 1.8

## 

### 

# Assemble working

This section will describe the abilities of the assembler.

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 exsis – 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:

* + - 1. No comment exists in the file.
      2. 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.

# Known Limitations-improvements

The following list contains the known limitation of the current version of the simulator.

* + 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 hexdecimal base, instead of a number.

# Manual

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

## Execution of Gui

|  |  |
| --- | --- |
| * + 1. Windows | 1. Extract the files to working directory. 2. Execute the Gui.bat |
| * + 1. Linux | 1. Extract the file to local folder. 2. Open the shell to this folder 3. Execute the following java command: 4. java -cp SimEasy8-jar-with-dependencies.jar meirdev.simulator.gui.frmae.GuiSimulator |

## Execution of assembler

|  |  |
| --- | --- |
| * + 1. Windows | 1. Extract the files to working directory. 2. Execute the Show.bat |
| * + 1. Linux | 1. Extract the file to local folder. 2. Open the shell to this folder 3. Execute the following java command: 4. 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.

# Gui

## In this section the Gui will be described. The ways to do every option.

## All parts view



Figure 1- Simulator parts

## File types used in the simulator

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

### 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.

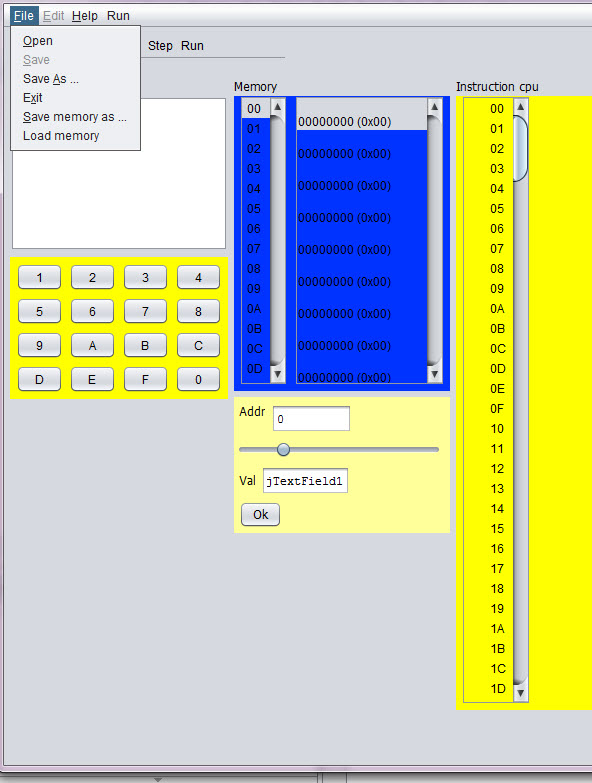
The format of the file is ascii, separated by colon

## Editing/saving assembler/memory files

- Edition frame to edit programs in assembly language.

### Open assembler file

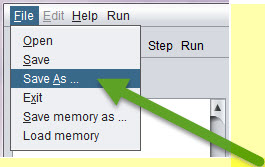
The file will be opened as in the following image, using the ***File/Open*** menu item /Sub menu item. Only ***asm*** files will be viewed from this dialog window.



2Open file

### Save assembler file

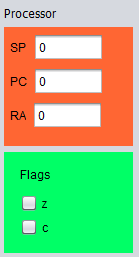
Save file – Use the ***Save*** / ***Save as*** sub menu. Extention of the file name could be ignored in this activity.



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.



7Processor registers and outcome indicators.

## Memory view

### Memory

### Instruction cpu

### Stack

- Some framesto display the computer's memory (256 bytes). This allows the student displayed at the same time the bottom of memory where the program into machine code is stored, 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)



### 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.

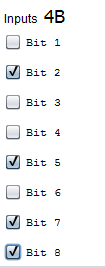
From the high bit at the top to the lower bit .

For example:

|  |  |
| --- | --- |
| Figure 8 battery of 8 switches - 4B value | Figure 9 0F value |

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).



## 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.[[2]](#footnote-5)

STOP button

* 1. Gui Menu

Reset menu – all the reset options.

# Gui behaviour

Behavior of the different elements:

-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.

## How to change memory value?

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

* 1. Click on the selected memory value
     1. One click – change the selected memory value, and change also the selected address in the Yellow qube
     2. 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.
  2. 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.

# Debugger

This section describes the options of th exection/debugger of the Simulator.

## Step Button

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

## Breakpoints

The user could add break point in the required address.

*Note:*Breakpoint could be added only after finished of assemble 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 instruction or press the STOP button.

## – Stop Button

Stops program execution.

# – Easy8 instructions list

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

* When command ends with the char: ‘I’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



# 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.

## Assembler program with IO

### 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:

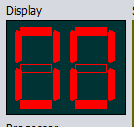


Figure 11Seven digit contain default value

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:
   1. OUT 77
   2. STOP

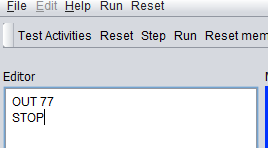


Figure 12Editie -ASM lines to display 77 in the seven digit

1. Assemble the file.
2. Run the assembled code.

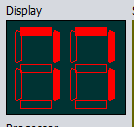


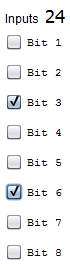
Figure 13Seven digit in the ending of the execution - hex 77

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

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

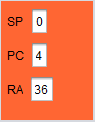
The Battery switch list should be as in the following image:

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

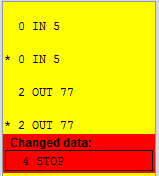


1. Clear the contents of the editor
2. Add the following lines:
   1. IN 5
   2. OUT 77
   3. STOP
3. Assemble the file.
4. Run the assembled code.

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

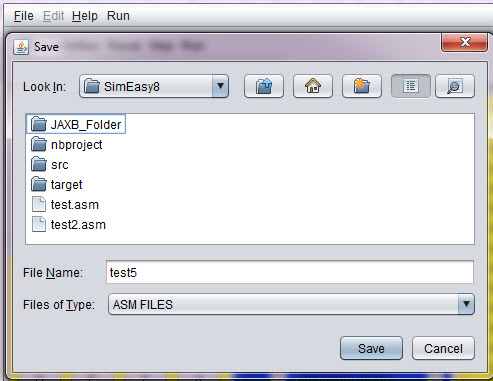
 

The instruction list will be as in the folowing image

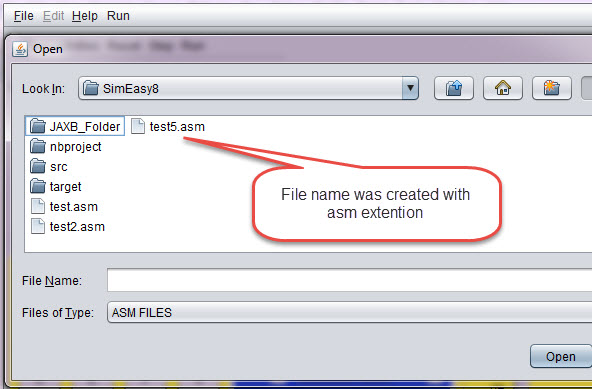


## Save file flow

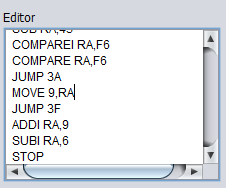
Saving the file name with no extension, will save the file with the correct extension, as in the following pictures. The result is – that open file will view the file and file will be opened with no issues:



4save asm file - test5 - with no extension



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



6Editor window

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

# 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

[Figure 1Seven digit contain default value - 00](#Toc453325212) 30

[Figure 2Editie -ASM lines to display 77 in the seven digit](#Toc453325213) 30

[Figure 3Seven digit in the ending of the execution - hex 77](#Toc453325214) 30

[Figure 4Seven digit contain default value - 00](#Toc453325215) 31

[Figure 5Seven digit in the ending of the execution - hex 77](#Toc453325216) 31

[Figure 6- Simulator parts](#Toc453325217) 18

[7Open file](#Toc453325218) 19

[8Save as - menu item q Sub menu item](#Toc453325219) 19

[9save asm file - test5 - with no extension](#Toc453325220) 20

[10The file was saved corectly - as test5.asm[Viewdlated by the open file ]](#Toc453325221) 20

[11Editor window](#Toc453325222) 21

[12Processor registers and outcome indicators.](#Toc453325223) 21

[Figure 13 battery of 8 switches - 4B value](#Toc453325224) 24

[Figure 14 0F value](#Toc453325225) 24

[Figure 15 battery of 8 switches - 4B value](#Toc453325226) 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

\*/

1. 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] [↑](#footnote-ref-2)
2. The reset menu – do not have this option. It’ [↑](#footnote-ref-5)