**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. 3

2. Introducción 10

The study of the Instruction Set Architecture is a very important subject in studies of computer design and programing. 10

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. 10

3. Objective 11

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. 11

It should be easy to use by the student. 11

It should be possible to use it from many platforms. 11

Also, we built it with java, so it could be used in window/linux/unix, and must include a graphical interface. 11

4. Requirements 12

4.1. List of requirements 12

i. Execution from windows/linux/unix. 12

ii. Execution of assembler on basic assembler language with minimal set of instruction list. 12

iii. Work with ASM files, assemble them and reload them from the memory. 12

iv. Ability to stop/continue execution of the system. Ability to add Breakpoints. 12

v. Ability to have input/output to display/external system. User interface to change memory. 12

vi. Load/save memory for working again on the same system. 12

vii. Show to the user the impact of the memory. 12

viii. Working with hexadecimal base 12

5. Application design 13

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. 13

5.1. Used design patterns 13

5.1.1. Singleton pattern 13

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. 13

5.1.2. Factory Pattern 13

to create the implementation of ActivityPiece in run time, for every step while execution the assembled code. Factory class name is ActivityPieceFactory. 13

5.2. Pseudo codes 13

5.2.1. Run/Step execution 13

#nextAddress = getNextAddress() 13

#step() 13

#updatePc() 13

#Show impact memory fields 13

#Show next step on the Instruction CPU window 13

5.2.2. Instruction implementation 13

5.3. Seven digit display 14

5.3.1.1. Explanations 14

5.4. Customization of classes 14

5.5. Implementations 14

5.5.1. Seven digit 14

5.5.1.1. High level design: 14

5.5.1.2. UML class diagram of class 14

5.5.1.3. Methods 14

5.5.1.4. Interface: 14

6. Known Limitations-improvements that could be done 16

6.1. Error handling of asm file 16

The assemble 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. 16

6.2. Seven digit update 16

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. 16

7. Manual 17

7.1. Execution of Gui 17

1. Windows 17

2. Linux 17

7.2. 17

7.2. Execution of assembler 17

1. Windows 17

2. Linux 17

7.3. 17

8. Gui 18

8.1. Parts view 18

8.2. File types used in the simulator 18

8.3. Optional files 18

8.3.1. ASM 18

8.3.1.1. This type of file contains the assembler code. 18

***8.3.2.*** MEM 18

8.3.2.1. This type of file contains the memory code. 18

8.4. Editing/saving assembler 18

8.5. Open assembler file 18

8.6. Save assembler file 19

8.6.1. Save file flow: 19

8.7. Memory view 22

8.7.1. Memory 22

8.7.2. Instruction cpu 22

8.7.3. Stack 22

8.8. Seven segment display 22

8.9. Input battery of 8 switches 23

8.10. Toolbar 24

8.11. Gui Menu 25

9. Gui behaviour 25

10. Debugger 26

10.1. - Step Button 26

10.2. - Breakpoint 26

10.3. - Run Button 26

10.4. - Stop Button 26

11. Easy8 Instruction Set 27

**11.1.** EASY8 INSTRUCTION SET. 27

12. Instruction 27

13. Operation code 27

14. Description 27

15. MOVEI RA, VALUE 27

18. MOVE RA, [ADDRESS] 27

21. MOVE [ADDRESS], RA 27

24. ADDI RA, VALUE 28

39. DEC RA 28

44. COMPARE RA, [ADDRESS] 28

56. JEQUAL ADDRESS 29

62. POP RA 29

67. RET 29

11.2. 29

12. 29

13. Appndix 30

13.1. Assembler file with IO 30

13.1.1. Basic flow – output to the Seven digit 30

13.1.2. Basic flow – Input from the seven switches battery 30

75. CONCLUSIONS AND FUTURE WORK 32

Index 35

List of pictures 36

1. Introducción

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.

1. 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.

It should be possible to use it from many platforms.

Also, we built it with java, so it could be used in window/linux/unix, and must include a graphical interface.

1. Requirements
   1. List of requirements
2. Execution from windows/linux/unix.
3. Execution of assembler on basic assembler language with minimal set of instruction list.
4. Work with ASM files, assemble them and reload them from the memory.
5. Ability to stop/continue execution of the system.  
   Ability to add Breakpoints.
6. Ability to have input/output to display/external system.  
   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 hexadecimal base

Edit with hex decimal values.

1. 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.

***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.***

* 1. Used design patterns
     1. 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.

* + 1. 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)

* 1. Pseudo codes
     1. Run/Step execution

#nextAddress = getNextAddress()

#step()

#updatePc()

#Show impact memory fields

#Show next step on the Instruction CPU window

* + 1. Instruction implementation

The implementation of one instruction is built from the following parts:

#**updateModel()**

**#create new event for gui**

**#Create jump event**

* 1. Seven digit display

**# FOR INT numOfDigits <=2**

**# FOR int pieceNumber <=7**

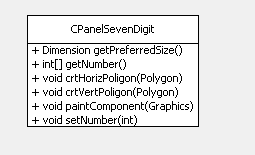
**# Draw the piece ON/OFF [using the graphics.fillPolygon method]**

* + - 1. 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.
   1. Customization of classes
3. JPanel – customized for display the Seven Digit
   1. Implementations
      1. Seven digit
         1. 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.

* + - 1. UML class diagram of class



* + - 1. Methods

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

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

* + - 1. Interface:

Creation – as in the following line, in the generated code:[[2]](#footnote-3)

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

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

1. Known Limitations-improvements that could be done
   1. Error handling of asm file

The assemble 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.

* 1. 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.

1. Manual
   1. 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 |

* 2. 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.

1. Gui
   1. Parts view



Figure 1- Simulator parts

* 1. File types used in the simulator

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

* 1. Optional files
     1. 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

* + 1. MEM

This type of file contains the memory code.

The format of the file is ascii, separated by colon

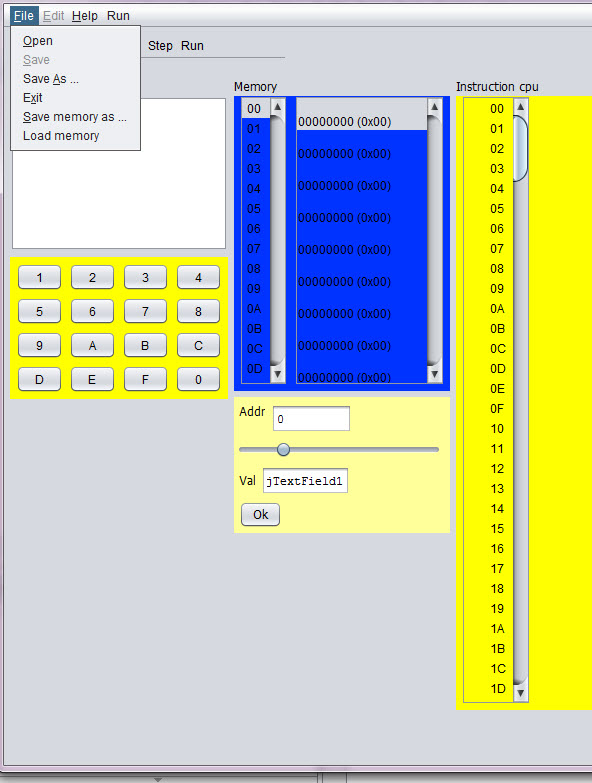
The graphic interface must include the following elements:

* 1. Editing/saving assembler

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

* 1. Open assembler file

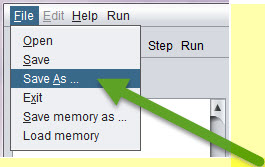
The file will be opened as in the following image, using the ***File/Open*** menu item /Sub menu item. Only ***asm*** files will be viewed from this dialog window.



2Open file

* 1. Save assembler file

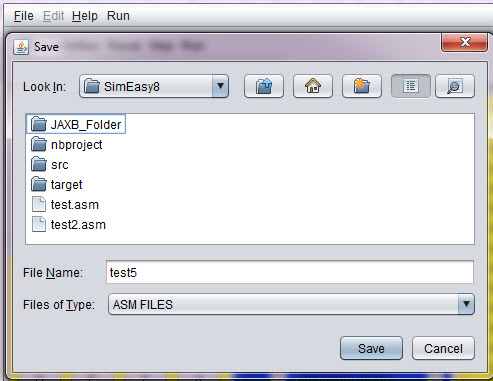
2. Save file – Use the ***Save*** / ***Save as*** sub menu. Extention of the file name could be ignored in this activity.



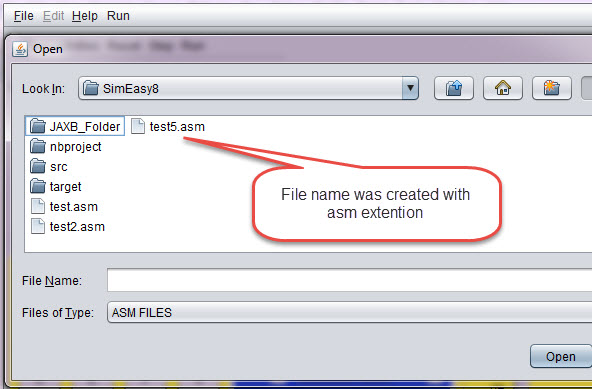
3Save as - menu item q Sub menu item

* + 1. Save file flow:

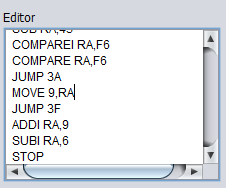
Saving the file name with no extention, will save the file with the correct extention, as in the following pictures. The result is – that open file will view the file and file will be opened with no issues:



4save asm file - test5 - with no extension



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

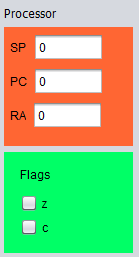


6Editor window

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

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.



7Processor registers and outcome indicators.

* 1. Memory view
     1. Memory
     2. Instruction cpu
     3. 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 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.

* 1. 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)



* 1. 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.

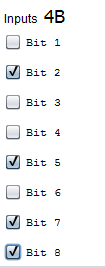
From the high bit at the top to the lower bit .

For example:

|  |  |
| --- | --- |
| Figure 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).



* 1. 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.[[3]](#footnote-4)

STOP button

* 1. Gui Menu

Reset menu – all the reset options.

1. 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.

1. Debugger
   1. - Step Button

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

* 1. - Breakpoint

The user could add break point in the required address.

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



* 1. - 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.

* 1. - 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.

1. Easy8 Instruction Set
   1. EASY8 INSTRUCTION SET.

|  |  |  |
| --- | --- | --- |
| 1. Instruction | 1. Operation code | 1. Description |
| 1. MOVEI RA, VALUE |  |  |
| 1. MOVE RA, [ADDRESS] |  |  |
| 1. MOVE [ADDRESS], RA |  |  |
| 1. ADDI RA, VALUE |  |  |
| ADD RA, [ADDRESS] |  |  |
| SUBI RA,VALUE |  |  |
| SUB RA, [ADDRESS] |  |  |
| INC RA |  |  |
| 1. DEC RA |  |  |
| COMPAREI RA, VALUE |  |  |
| 1. COMPARE RA, [ADDRESS] |  |  |
| JUMP ADDRESS |  |  |
| JLESS ADDRESS |  |  |
| JGREATER ADDRESS |  |  |
| 1. JEQUAL ADDRESS |  |  |
| PUSH RA |  |  |
| 1. POP RA |  |  |
| CALL ADDRES |  |  |
| 1. RET |  |  |
|  |  |  |




2. Appndix
   1. Assembler file with IO
      1. 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:

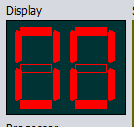


Figure 11Seven digit contain default value

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

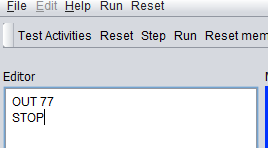


Figure 12Editie -ASM lines to display 77 in the seven digit

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

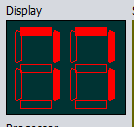


Figure 13Seven digit in the ending of the execution - hex 77

* + 1. 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:

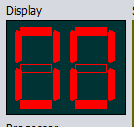


Figure 14Seven digit contain default value - 00

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:
   1. IN
   2. OUT RA
   3. STOP
5. Assemble the file.
6. Run the assembled code.

Figure 15Seven digit in the ending of the execution - hex 77

1. 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

asm

Error handling, 16

ASM

Using, 12, 21, 30, 31, 36

assemble

Breakpoint, 26

assumption, 13

Factory, ***13***

File format

ASM, 18

MEM, 18

**fillPolygon**

Using API, 14

hexdecimal

Diplay in seven digit, 16

input/output, ***12***

JPanel, ***13***, ***14***

manual, 17, *See* manual

paintComponent. *See* Pseudo codes

param

console. *See Console*

Pseudo codes, 13

seven-segment, 22, *See*

Singleton

Using, ***13***

switches battery

Input, 30

Windows

execution, ***17***

execution of Gui, ***17***

working directory

Using, ***17***

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. NetBeans add the customized creation code. The component has ***Custom creation code*** in the Code properties tab. [↑](#footnote-ref-3)
3. The reset menu – do not have this option. It’ [↑](#footnote-ref-4)