uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2018-2019

Trabajo Fin de Grado

Diseño y desarrollo de un simulador genérico para programación en ensamblador

---

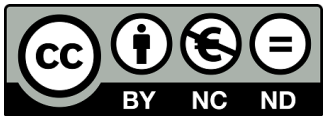
Diego Camarmas Alonso

Tutor

Félix García Carballeira

Leganés, Madrid, España

Julio 2019



Si no puedes volar entonces corre, si no puedes correr entonces camina, si no puedes caminar entonces arrástrate, pero sea lo que sea que hagas, sigue moviéndote hacia adelante

Martin Luthr King, Jr

Diseño y desarrollo de un simulador genérico para programación en ensamblador

por

Diego Camarmas Alonso

Resumen

CREATOR es un simulador docente genérico para programar en ensamblador en el que se puede simular el funcionamiento de diferentes arquitecturas sobre la misma herramienta. Este simulador está diseñado para ser utilizado como una herramienta en la que los alumnos pueden poner en práctica los cocimientos vistos en las clases teóricas de las asignaturas de Arquitectura y Estructura de Computadores.

En esta herramienta los usuarios pueden modificar las principales características que tiene la arquitectura de un computador como puede ser el conjunto de instrucciones que ejecuta o el conjunto de registros que dispone. Además, durante la ejecución podrán ver el estado en el que se encuentra el computador, es decir, el valor que almacenan los registros y la memoria. Otra funcionalidad destacada que ofrece esta herramienta es la posibilidad de generar bibliotecas de funciones y también poder utilizar en un programa en ensamblador bibliotecas externas. Como se trata de una aplicación web permite que CREATOR pueda ser utilizado en cualquier situación y dispositivo.

Debido a todas las características que se han mencionado anteriormente con este simulador se persigue hacer la enseñanza y aprendizaje de estas asignaturas mucho más fácil para los alumnos.

Palabras clave: CREATOR, Ensamblador, Genérico, Simulador

Tutor: Félix García Carballeira
Catedrático de Universidad

Design and development of a generic simulator for assembly programming.

by

Diego Camarmas Alonso

Abstract

CREATOR is a generic teaching simulator to program in assembly in which you can simulate the operation of different architectures on the same tool. This simulator is designed to be used as a tool in which students can put into practice the brews seen in the theoretical classes of the subjects of Architecture and Computer Structure.

In this tool, users can modify the main characteristics of a computer's architecture, such as the set of instructions it executes or the set of registers it has. In addition, during the execution, they will be able to see the state in which the computer is, that is to say, the value that the registers and the memory store. Another outstanding functionality offered by this tool is the possibility of generating function libraries and also being able to use external libraries in an assembly program. As it is a web application, CREATOR can be used in any situation and device.

Due to all the features that have been mentioned previously, with this simulator is intended to make teaching and learning of these subjects much easier for students.

Keywords: CREATOR, Assembly, Generic, Simulator

Supervisor: Félix García Carballeira
Full Professor

Agradecimientos

En este apartado me gustaría dedicar unas palabras a todas aquellas personas que me han estado apoyando a lo largo de esta etapa de mi vida.

En primer lugar a mis padres, Paloma y Félix, que a lo largo de toda mi vida y, en especial, durante estos últimos años que me han estado apoyando y aconsejando para que realizase las cosas que más feliz me hacían, al igual que toda mi familia que también habéis sido un gran apoyo.

En segundo lugar quiero agradecer a mi tutor Félix la confianza depositada en mi para realizar este proyecto que ha sido muy interesante y me ha permitido aplicar muchos de los conocimientos aprendidos durante estos años, además de aprender otros muchos.

También quiero dedicar unas palabras a Alejandro Calderón, gracias por los consejos que me has dado a lo largo de este proyecto y la confianza que también depositaste en mi.

No me puedo terminar de escribir estas líneas sin dedicar unas palabras a mis amigos de Aranjuez que aunque durante estos últimos años hemos pasado menos tiempo juntos siempre hemos buscado un momento para vernos y pasar buenos ratos juntos.

También quiero acordarme de mis amigos de la universidad, mis compañeros de fatigas, por aguantarme todos los días en clase y en la biblioteca mientras hacíamos prácticas. En especial quiero acordarme de Luis, que durante todos estos años ha sido un buen compañero de prácticas, pero sobretodo un gran amigo. Para finalizar, también tengo que mencionar a Elías, una persona que era capaz de sacarme de quicio algunas veces, pero que también me ha ayudado mucho durante estos años.

A todos vosotros, muchas gracias.

Índice general

1 Introducción **1**

1.1 Motivación 1

1.2 Objetivos 3

1.3 Estructura del documento 3

2 Estado del arte **5**

2.1 Simuladores de lenguaje ensamblador 5

2.1.1 SPIM 5

2.1.2 MARS 7

2.1.3 WebMIPS 9

2.1.4 VisUAL 10

2.1.5 ARMSim 12

2.1.6 Comparativa de los simuladores 14

2.2 Tecnologías web 15

2.2.1 HTML5 15

2.2.2 Frameworks 17

2.3 Entornos de desarrollo de aplicaciones móviles 19

2.3.1 Apache Cordova 19

2.3.2 Ionic 21

3 Análisis **23**

3.1 Descripción del proyecto 23

3.2 Requisitos 24

3.2.1 Requisitos de usuario 26

3.2.2 Modelo de casos de uso 37

3.2.3 Requisitos software 49

3.2.4 Matriz de trazabilidad de requisitos 67

3.3 Marco regulador 69

3.3.1 Licencias de uso del software 69

3.3.2 Aspectos legales 69

4 Diseño **71**

4.1 Estudio de la solución final 71

4.1.1 Solución final 74

4.2 Arquitectura del simulador 75

4.2.1 Editor de arquitectura 76

4.2.2 Compilador 81

4.2.3 Kernel de simulación 82

4.2.4 Interfaz de usuario 82

5 Implementación y despliegue **85**

5.1 Implementación 85

5.2 Despliegue 99

6 Verificación, validación y evaluación **101**

6.1 Verificación y validación 101

6.1.1 Pruebas de verificación 102

6.1.2 Pruebas de validación 122
6.2 Evaluación 139

7 Planificación y presupuesto **145**
7.1 Planificación 145
7.1.1 Metodología 145
7.1.2 Ciclo de vida 146
7.1.3 Tiempo estimado 147
7.2 Presupuesto 149
7.2.1 Coste del proyecto 149
7.2.2 Oferta del proyecto 152
7.3 Entorno socio-económico 152

8 Conclusiones y trabajos futuros **155**
8.1 Conclusiones 155
8.2 Contribuciones 157
8.3 Trabajos futuros 158

Anexo A - Manual de usuario **159**
A Pasos para ejecutar en CREATOR 159
B Carga de la arquitectura a utilizar 159
C Gestión del código ensamblador 161
D Simulación 162

Appendix B - Extended abstract **167**
A Introduction 167
A.1 Motivation 167
A.2 Objectives 168

B Design 169

B.1 Final solution 169

B.2 Simulator Architecture 170

C Evaluation 172

D Conclusions and future work 176

D.1 Conclusions 176

D.2 Future work 177

Glosario **179**

Siglas **181**

Bibliografía **183**

Índice de figuras

2.1 Interfaz de SPIM 6

2.2 Interfaz de QtSpim 7

2.3 Interfaz de MARS 9

2.4 Interfaz de WebMIPS 10

2.5 Interfaz de VisUAL 12

2.6 Interacción de ARMSim y QtARMSim 13

2.7 Interfaz de QtARMSim 14

2.8 Arquitectura de una aplicación creada con Apache Cordova 21

3.1 Casos de uso de la gestión de la arquitectura 38

3.2 Casos de uso de la gestión del código ensamblador 38

3.3 Casos de uso de la simulación 39

4.1 Arquitectura de la aplicación 76

4.2 Estructura de una instrucción 79

5.1 Estructura de ficheros 86

6.1 Evaluación 1 CREATOR 140

6.2 Evaluación 1 MARS 140

6.3 Evaluación 2 CREATOR 141

6.4 Evaluación 2 MARS 141

6.5 Evaluación 3 CREATOR 141
6.6 Evaluación 3 MARS 141
6.7 Evaluación 4 CREATOR 142
6.8 Evaluación 4 MARS 142
6.9 Evaluación 5 CREATOR 143
6.10 Evaluación 5 MARS 143

7.1 Diagrama de Gantt 148

B.1 Carga de una arquitectura proporcionada 160
B.2 Formulario de carga de una nueva arquitectura 160
B.3 Carga de la nueva arquitectura 161
C.1 Selección del botón “Assembly” 161
C.2 Pantalla Assembly 162
D.1 Pantalla principal 163
D.2 Pantalla principal: visualización de la memoria 163
D.3 Pantalla principal: visualización de las estadísticas 164
D.4 Pantalla principal: visualización de la calculadora 164
D.5 Pantalla principal: visualización de una ejecución 165
B.1 Application Architecture 171
C.1 Evaluation 1 CREATOR 173
C.2 Evaluation 1 MARS 173
C.3 Evaluation 2 CREATOR 173
C.4 Evaluation 2 MARS 174
C.5 Evaluation 3 CREATOR 174
C.6 Evaluation 3 MARS 174
C.7 Evaluation 4 CREATOR 174

C.8 Evaluation 4 MARS 175

C.9 Evaluation 5 CREATOR 175

C.10 Evaluation 5 MARS 175

Índice de tablas

2.1 Comparativa de simuladores 15

3.1 Plantilla de la especificación de requisitos 26

3.2 Requisito de usuario RU-C01 27

3.3 Requisito de usuario RU-C02 27

3.4 Requisito de usuario RU-C03 28

3.5 Requisito de usuario RU-C04 28

3.6 Requisito de usuario RU-C05 28

3.7 Requisito de usuario RU-C06 29

3.8 Requisito de usuario RU-C07 29

3.9 Requisito de usuario RU-C08 29

3.10 Requisito de usuario RU-C09 30

3.11 Requisito de usuario RU-C10 30

3.12 Requisito de usuario RU-C11 31

3.13 Requisito de usuario RU-C12 31

3.14 Requisito de usuario RU-C13 31

3.15 Requisito de usuario RU-C14 32

3.16 Requisito de usuario RU-C15 32

3.17 Requisito de usuario RU-C16 32

3.18 Requisito de usuario RU-C17 33

3.19 Requisito de usuario RU-C18 33
3.20 Requisito de usuario RU-R01 34
3.21 Requisito de usuario RU-R02 34
3.22 Requisito de usuario RU-R03 35
3.23 Requisito de usuario RU-R04 35
3.24 Requisito de usuario RU-R05 35
3.25 Requisito de usuario RU-R06 36
3.26 Requisito de usuario RU-R07 36
3.27 Requisito de usuario RU-R08 36
3.28 Requisito de usuario RU-R09 37
3.29 Requisito de usuario RU-R10 37
3.30 Plantilla para los casos de uso 39
3.31 Caso de uso CU-01 40
3.32 Caso de uso CU-02 40
3.33 Caso de uso CU-03 41
3.34 Caso de uso CU-04 41
3.35 Caso de uso CU-05 42
3.36 Caso de uso CU-06 42
3.37 Caso de uso CU-07 43
3.38 Caso de uso CU-08 43
3.39 Caso de uso CU-09 44
3.40 Caso de uso CU-10 44
3.41 Caso de uso CU-11 45
3.42 Caso de uso CU-12 45
3.43 Caso de uso CU-13 46

3.44 Caso de uso CU-14 46
3.45 Caso de uso CU-15 47
3.46 Caso de uso CU-16 47
3.47 Caso de uso CU-17 48
3.48 Caso de uso CU-18 48
3.49 Caso de uso CU-19 49
3.50 Requisito software RW-F-F01 49
3.51 Requisito software RW-F-F02 50
3.52 Requisito software RW-F-F03 50
3.53 Requisito software RW-F-F04 50
3.54 Requisito software RW-F-F05 51
3.55 Requisito software RW-F-F06 51
3.56 Requisito software RW-F-F07 51
3.57 Requisito software RW-F-F08 52
3.58 Requisito software RW-F-F09 52
3.59 Requisito software RW-F-F10 52
3.60 Requisito software RW-F-F11 53
3.61 Requisito software RW-F-F12 53
3.62 Requisito software RW-F-F13 53
3.63 Requisito software RW-F-F14 54
3.64 Requisito software RW-F-F15 54
3.65 Requisito software RW-F-F16 54
3.66 Requisito software RW-F-F17 55
3.67 Requisito software RW-F-F18 55
3.68 Requisito software RW-F-F19 55

3.69 Requisito software RW-F-F20 56
3.70 Requisito software RW-F-F21 56
3.71 Requisito software RW-F-F22 56
3.72 Requisito software RW-F-F23 57
3.73 Requisito software RW-F-F24 57
3.74 Requisito software RW-F-F25 57
3.75 Requisito software RW-F-F26 58
3.76 Requisito software RW-F-F27 58
3.77 Requisito software RW-F-F28 58
3.78 Requisito software RW-F-F29 59
3.79 Requisito software RW-F-F30 59
3.80 Requisito software RW-F-F31 59
3.81 Requisito software RW-F-F32 60
3.82 Requisito software RW-F-F33 60
3.83 Requisito software RW-F-F34 60
3.84 Requisito software RW-F-F35 61
3.85 Requisito software RW-F-F36 61
3.86 Requisito software RW-F-F37 61
3.87 Requisito software RW-F-F38 62
3.88 Requisito software RW-F-F39 62
3.89 Requisito software RW-F-F40 62
3.90 Requisito software RW-F-I01 63
3.91 Requisito software RW-F-I02 63
3.92 Requisito software RW-NF-PL01 63
3.93 Requisito software RW-NF-PL02 64

3.94 Requisito software RW-NF-PL03 64
3.95 Requisito software RW-NF-PL04 64
3.96 Requisito software RW-NF-PL05 65
3.97 Requisito software RW-NF-PL06 65
3.98 Requisito software RW-NF-PL07 65
3.99 Requisito software RW-NF-PL08 66
3.100 Requisito software RW-NF-PL09 66
3.101 Requisito software RW-NF-UI01 66
3.102 Requisito software RW-NF-UI02 67
3.103 Requisito software RW-NF-R01 67
3.104 Matriz de trazabilidad de requisitos 68

4.1 Comparativa de frameworks 73

6.1 Plantilla pruebas de verificación 102
6.2 Prueba de verificación PVE-01 103
6.3 Prueba de verificación PVE-02 103
6.4 Prueba de verificación PVE-03 104
6.5 Prueba de verificación PVE-04 105
6.6 Prueba de verificación PVE-05 106
6.7 Prueba de verificación PVE-06 107
6.8 Prueba de verificación PVE-07 108
6.9 Prueba de verificación PVE-08 108
6.10 Prueba de verificación PVE-09 109
6.11 Prueba de verificación PVE-10 110
6.12 Prueba de verificación PVE-11 111
6.13 Prueba de verificación PVE-12 112

6.14 Prueba de verificación PVE-13 113
6.15 Prueba de verificación PVE-14 113
6.16 Prueba de verificación PVE-15 114
6.17 Prueba de verificación PVE-16 115
6.18 Prueba de verificación PVE-17 116
6.19 Prueba de verificación PVE-18 116
6.20 Prueba de verificación PVE-19 117
6.21 Prueba de verificación PVE-20 118
6.22 Prueba de verificación PVE-21 118
6.23 Prueba de verificación PVE-22 119
6.24 Prueba de verificación PVE-23 120
6.25 Matriz de trazabilidad pruebas de verificación 121
6.26 Plantilla pruebas de validación 122
6.27 Prueba de validación PVA-01 123
6.28 Prueba de validación PVA-02 123
6.29 Prueba de validación PVA-03 124
6.30 Prueba de validación PVA-04 125
6.31 Prueba de validación PVA-05 126
6.32 Prueba de validación PVA-06 127
6.33 Prueba de validación PVA-07 128
6.34 Prueba de validación PVA-08 129
6.35 Prueba de validación PVA-09 130
6.36 Prueba de validación PVA-10 131
6.37 Prueba de validación PVA-11 132
6.38 Prueba de validación PVA-12 132

6.39 Prueba de validación PVA-13 133
6.40 Prueba de validación PVA-14 134
6.41 Prueba de validación PVA-15 135
6.42 Prueba de validación PVA-16 135
6.43 Prueba de validación PVA-17 136
6.44 Prueba de validación PVA-18 137
6.45 Prueba de validación PVA-19 137
6.46 Prueba de validación PVA-20 138
6.47 Prueba de validación PVA-21 138
6.48 Matriz de trazabilidad pruebas de validación 139

7.1 Resumen del proyecto 149
7.2 Costes en recursos humanos 150
7.3 Costes en equipamiento 150
7.4 Costes de material fungible 151
7.5 Resumen de los costes 152
7.6 Resumen de los costes 152

Capítulo 1

Introducción

En este primer capítulo se va a presentar el proyecto desarrollado. En primer lugar, se va a indicar la motivación (Sección 1.1, Motivación), después, los objetivos que se quieren conseguir (Sección 1.2, Objetivos) y por último, la estructura del presente documento (Sección 1.3, Estructura del documento).

1.1. Motivación

En la Ingeniería Informática, una pieza fundamental es el conocer cómo es la arquitectura de un computador ya que permite comprender el funcionamiento de este a bajo nivel. Es por esto que su enseñanza debe ser lo más completa posible, pero, que a la vez, no suponga grandes retos a los alumnos a la hora de realizar los ejercicios prácticos consiguiendo así una comprensión mucho mayor del funcionamiento de un computador.

Uno de los principales retos a los que se deben enfrentar los profesores que imparten las asignaturas de Estructura y Arquitectura de Computadores a la hora de preparar las clases prácticas es elegir correctamente la herramienta que los alumnos van a utilizar, ya que esta herramienta tiene que simular el funcionamiento de un computador de la forma más parecida posible a lo impartido en las clases teóricas.

Actualmente, existen distintas herramientas con las que se pueden simular el funcionamiento interno de un computador, pero éstas son demasiado complejas de usar, poco intuitivas, no son accesibles por dispositivos móviles (*Smartphones* o *Tablets*), ya que la

gran mayoría están desarrolladas para ordenador, y suelen estar diseñadas para simular una única arquitectura.

Es por esto por lo que se ha decidido diseñar y desarrollar un simulador genérico para programar en ensamblador y que se llama CREATOR (didaCtic geneRic assEmbly progrAmming simulaTOR). Se pensó que este simulador debía ser genérico para poder simular diferentes tipos de arquitecturas con una misma herramienta y que permite a los alumnos estar más familiarizados con el funcionamiento de dicha herramienta y, por lo tanto, realizar las tareas de una forma más rápida y sencilla. Esta característica también permite que se puedan añadir nuevas arquitecturas posteriormente, lo que le da un valor añadido, ya que el mundo de la informática avanza a gran velocidad y la enseñanza debe ir al mismo compás. Por todo ello, se ha comenzado a desarrollar este proyecto que se realiza dentro de un proyecto de innovación docente de la Universidad Carlos III de Madrid durante el curso 2018 - 2019 denominado "Mejora de los recursos didácticos para la realización de prácticas en la asignatura Estructura de Computadores", que ha sido coordinado por Félix García Carballeira y que será utilizado en los próximos cursos en esta asignatura.

Esta herramienta aparte de poder simular diferentes arquitecturas también permite a los alumnos editar algunas de las características de la arquitectura en la que se está trabajando, como puede ser crear, modificar o eliminar registros, instrucciones ensamblador o pseudointrucciones.

Como se comentaba anteriormente, la mayoría de las herramientas disponibles actualmente para este propósito están desarrollados para ordenadores, pero, en la actualidad, cada vez existen más dispositivos móviles (*Smartphones* o *tablets*) por lo que se decidió que esta herramienta también se pudiera utilizar en dispositivos móviles y así no limitar su uso.

Por lo tanto, proponemos desarrollar un simulador que sea: simple, intuitivo, genérico y multiplataforma y en el que se pueda comprender el funcionamiento de un computador a bajo nivel gracias a la creación y ejecución de programas en diferentes lenguajes ensamblador y a la gran libertad de la que disponen los usuarios para editar la arquitectura sobre la que se está trabajando.

1.2. Objetivos

En este proyecto el objetivo principal es realizar un simulador genérico para programar en ensamblador y que permite ejecutar programas ensamblador sobre diferentes tipos de arquitecturas, a la vez de editar éstas. Esto permite que los alumnos puedan comprender el funcionamiento de diferentes tipos de arquitecturas de una forma más sencilla e interactiva, así como aprender a programar en lenguaje ensamblador.

Además del objetivo principal, el simulador debe permitir:

- Definir diferentes juegos de instrucciones. El usuario podrá definir la arquitectura, componentes y juego de instrucciones. El simulador debe ser lo suficiente genérico para poder definir cualquier juego de instrucciones.
- Escribir programas en lenguaje ensamblador utilizando los juegos de instrucciones disponibles.
- Compilar programas escritos en lenguaje ensamblador.
- Ejecutar programas en ensamblador y depurar su ejecución.
- Visualizar el estado del computador después de ejecutar cada una de las instrucciones que conforman un programa.
- Visualizar y editar los diferentes atributos de los componentes que forman la arquitectura.

1.3. Estructura del documento

Este documento constará de los siguientes capítulos:

- Capítulo 1. Introducción: en este capítulo se muestra un pequeño preámbulo del contenido de este documento. Además, también se explican las motivaciones y los objetivos de este proyecto.
- Capítulo 2. Estado del arte: en este apartado se analizan los distintos simuladores que existen en la actualidad, las distintas tecnologías web que se han usado en el

desarrollo de este proyecto, así como algunos entornos de desarrollo de aplicaciones móviles.

- Capítulo 3. Análisis: se detalla el proyecto, se define el conjunto de los requisitos seguidos y se indica el marco regulador del proyecto.
- Capítulo 4. Diseño: se describe el diseño del sistema creado, al igual que los diferentes componentes que lo forman.
- Capítulo 5. Implementación y despliegue: en este capítulo se detalla la implementación de las partes principales del software implementado. Además, también se explica cómo se va a realizar el despliegue del sistema.
- Capítulo 6. Verificación, validación y evaluación: se indica cómo se ha verificado el correcto funcionamiento del proyecto.
- Capítulo 7. Planificación y presupuesto: se explica la planificación seguida, se realiza un presupuesto del proyecto y se de detalla el entorno socioeconómico.
- Capítulo 8. Conclusiones y trabajos futuros: en este último capitulo se exponen las conclusiones a las que se llega tras la realización de este proyecto, así como los trabajos futuros.

Capítulo 2

Estado del arte

En este capítulo se va a presentar el estado del arte, en el que se detalla en qué etapa se encuentran las tecnologías utilizadas para el desarrollo de este proyecto. En primer lugar, se van a presentar diferentes simuladores de lenguaje ensamblador que existen en la actualidad (Sección 2.1). A continuación, se van a exponer las tecnologías web más usadas en la actualidad (Sección 2.2) y, por último, se van a explicar diferentes entornos de desarrollo de aplicaciones móviles (Sección 2.3).

2.1. Simuladores de lenguaje ensamblador

En este apartado se van a presentar diferentes simuladores de lenguaje ensamblador. Todos ellos están relacionados con el proyecto que se va a a desarrollar, ya que éstos, son usados para la enseñanza de las asignaturas de Arquitectura y Estructura de Computadores. Los simuladores que se van a exponer a continuación son: SPIM (Sección 2.1.1), MARS (Sección 2.1.2), WebMIPS (Sección 2.1.3), VisUAL (Sección 2.1.4) y ARMSim (Sección 2.1.5).

2.1.1. SPIM

SPIM [1] es un simulador que fue desarrollado por James Laurus en el año 1990. Este simulador, en un primer momento, era capaz de ejecutar programas escritos en el lenguaje ensamblador MIPS-I que es el que utilizan los procesadores MIPS R2000/R3000,

actualmente el simulador ejecuta la arquitectura MIPS32 que es la versión más reciente. Este simulador tiene versiones para los principales Sistemas Operativos: Microsoft Windows, Mac OS X y Linux.

Como se ha mencionado anteriormente, este simulador tiene la función de ejecutar programas ensamblador, ya que implementa la gran mayoría del conjunto de instrucciones que utiliza MIPS32 y las llamadas al sistema que éste puede producir, pero, además, también permite depurar este código para poder solventar los posibles errores que se hayan podido producir.

Sin embargo, SPIM, no puede ejecutar el conjunto completo de instrucciones de MIPS ya que no soporta direcciones de 64 bits que son las usadas en MIPS64. Adicionalmente, otra de las funcionalidades de las que carece SPIM es que no puede ejecutar un programa binario, que es el programa ensamblador ya compilado.

SPIM ofrece una interfaz basada en terminal y una interfaz de ventanas mediante QtSpim, que es la versión más reciente de SPIM y que tiene la particularidad de tener la misma interfaz de usuario para todos los Sistemas Operativos mencionados anteriormente. A continuación, se muestra en las siguientes Figuras la interfaz de SPIM seguida de la de QtSpim.

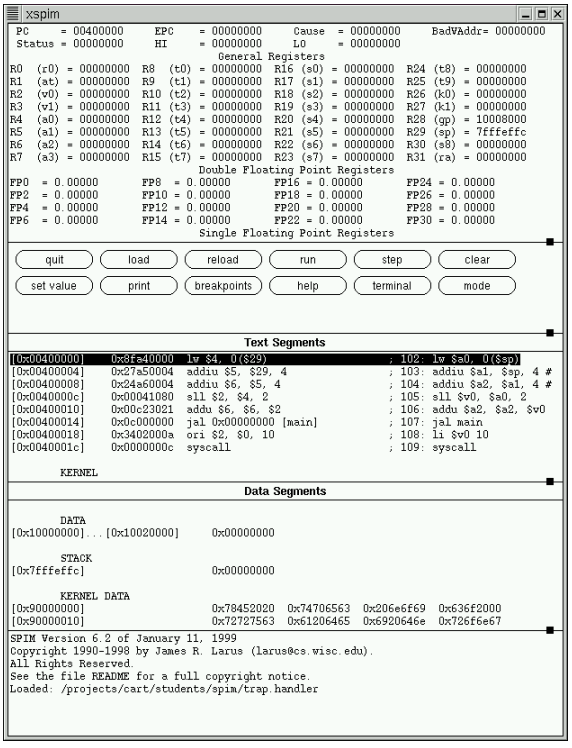


Fig. 2.1. Interfaz de SPIM

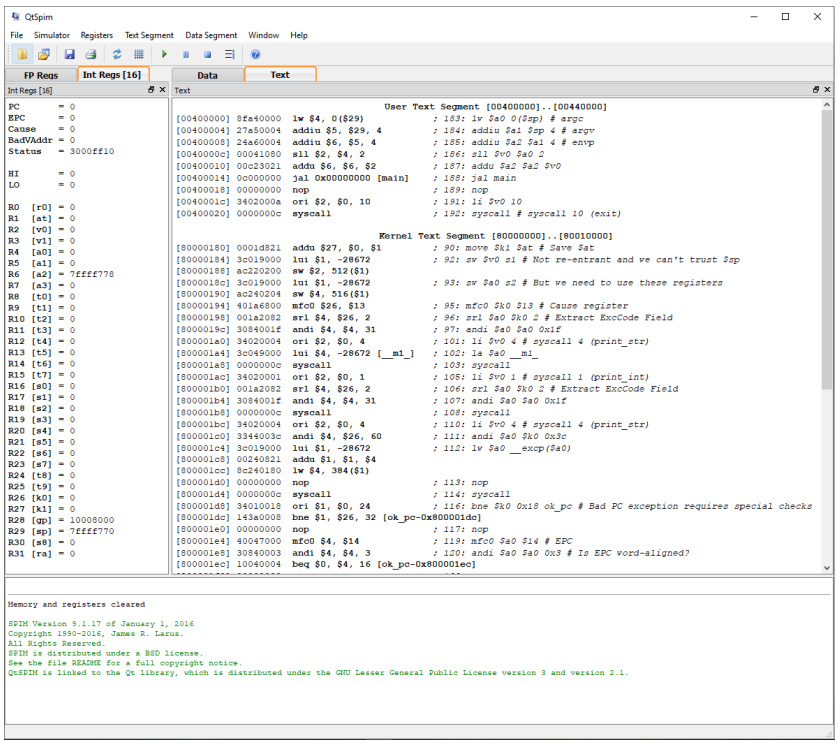


Fig. 2.2. Interfaz de QtSpim

2.1.2. MARS

MARS [2] es un simulador de programas escritos en el lenguaje ensamblador MIPS. Éste dispone de interfaz gráfica y está desarrollado en Java. El simulador fue diseñado para el ámbito educativo y para ofrecer una alternativa al simulador SPIM, explicado en la sección 2.1.1.

MARS está diseñado en base a la arquitectura MIPS RISC y el lenguaje ensamblador asociado y, por ello, ofrece un conjunto limitado de instrucciones. Las acciones que los usuarios pueden realizar con estas instrucciones pueden ser de entrada/salida, saltos, llamadas al sistema u operaciones aritmético-lógicas, entre otras. Algunas de las características que este simulador ofrece son:

- Control de la velocidad de ejecución (indicada en instrucciones por segundo).
- Permite visualizar treinta y dos registros a la vez y seleccionarlos.
- Ofrece la posibilidad de modificar los valores de los registros y la memoria mediante *"What You See Is What You Get"*(WYSIWYG).

- Los datos se pueden visualizar en hexadecimal o en decimal.
- Incluye un editor y un ensamblador como parte de su IDE.
- Permite ejecutar una instrucción de manera inversa.

Como se mencionó anteriormente, MARS se diseñó para ofrecer una alternativa a SPIM, en consecuencia, se van a exponer algunas de las mejoras que implementa MARS con respecto a SPIM:

- En SPIM, para modificar los valores de un registro o memoria, se tienen que efectuar varios pasos. Sin embargo, en MARS, esta acción se puede realizar de una forma más sencilla gracias a WYSIWYG.
- Para establecer un punto de interrupción en SPIM se tienen que realizar varios pasos para indicar la línea donde se desea poner, no obstante, MARS dispone de una casilla de verificación al lado de cada una de las líneas de código para asignar este punto de interrupción.
- SPIM puede ejecutar un programa instrucción a instrucción o de forma ininterrumpida. MARS ofrece estas dos opciones y, además, también permite indicar la velocidad de ejecución en instrucciones por segundo.
- SPIM no incluye un editor integrado teniendo que hacerse uso de editores externos para escribir el código o editarlo. Por otro lado, MARS ofrece un editor integrado que permite realizar estas funciones sin tener que recurrir a un editor externo.
- MARS permite ejecutar una instrucción en orden inverso, es decir, puede volver al estado anterior del programa, mientras que SPIM no dispone de esta funcionalidad.

Para finalizar, se muestra en la siguiente Figura la interfaz de usuario de MARS.

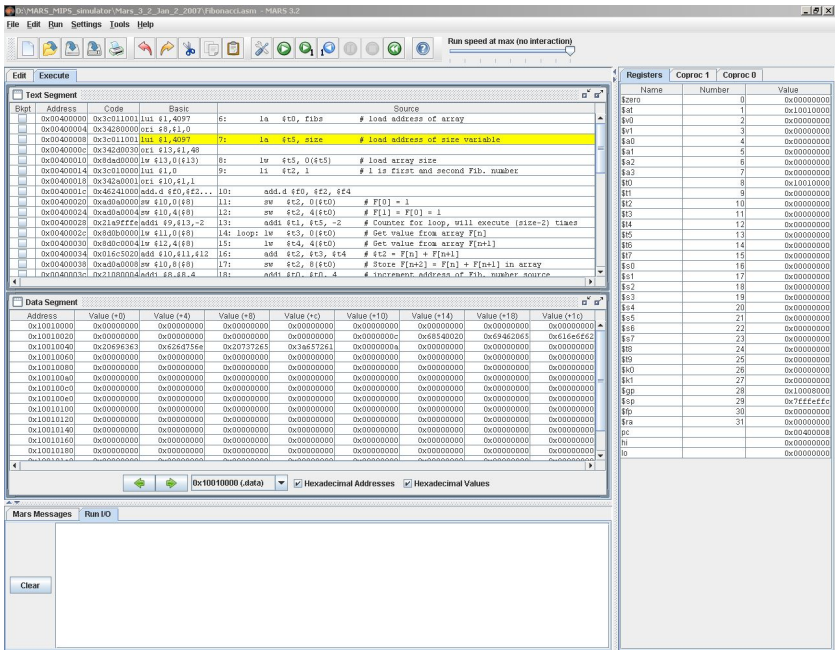


Fig. 2.3. Interfaz de MARS

2.1.3. WebMIPS

WebMIPS [3] es un simulador que crearon los profesores de la asignatura “Arquitectura de Computadores” de la Universidad de Siena, en Italia, para que los alumnos tuvieran una herramienta de simulación acorde a los conocimientos que se impartían en clase. Este simulador se desarrolló para ser usado mediante una página web que permite a los usuarios usar esta herramienta en cualquier Sistema Operativo, a parte de no tener que instalarla en su ordenador. Por eso se utilizó el lenguaje ASP.NET [4].

Este simulador, a diferencia de los simuladores explicados anteriormente, ofrece un diagrama en el que se puede observar un *pipeline* que tiene cinco etapas con las que se puede mostrar el estado de la gran mayoría del hardware de las arquitecturas MIPS. Sin embargo, este simulador no es capaz de simular todo el conjunto de instrucciones que ofrece MIPS. Solo se puede simular un conjunto limitado, que fue determinado por sus creadores, para que el simulador fuera lo más parecido posible a lo impartido en clase. Las características más relevantes de este simulador son:

- WebMIPS es capaz de detectar errores en el código ensamblador que el usuario introduce.

- El simulador pone a disposición del usuario ejemplos de programas ensamblador.
- Esta herramienta es capaz de ejecutar un programa ensamblador paso a paso o todo el programa.
- Cuando el programa se ejecuta paso a paso, WebMIPS ofrece al usuario la posibilidad de ver el avance de las instrucciones en cada sección del *pipeline*.
- Cuando se completa la ejecución de un programa, podemos ver el número total de ciclos de reloj que han sido necesarios para su ejecución.

Para terminar, se muestra una imagen donde se puede ver la interfaz de usuario de WebMIPS, en la que destaca el diagrama en el que se muestra el *pipeline* de las instrucciones.

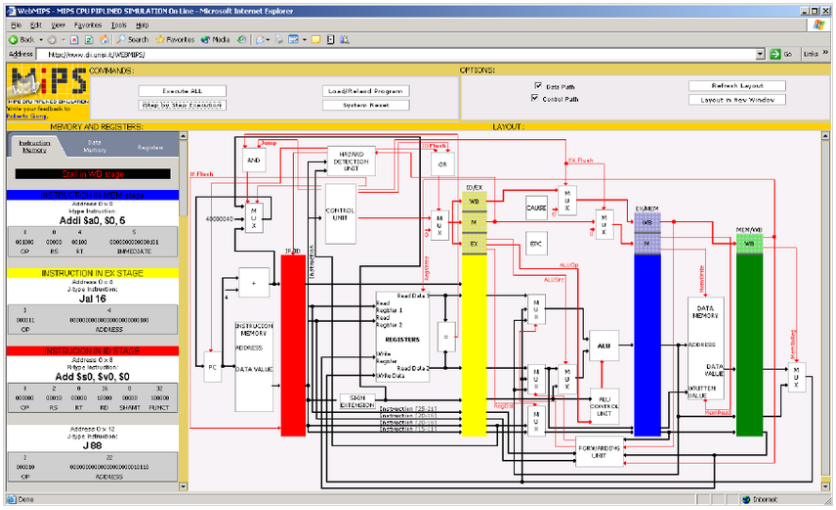


Fig. 2.4. Interfaz de WebMIPS

2.1.4. VisUAL

VisUAL [5] es un simulador del lenguaje ensamblador ARM. Éste fue diseñado para utilizarse en el "Curso de Introducción a la Arquitectura de Computadores" del *Imperial College* de Londres. Esta herramienta es multiplataforma porque ofrece versiones para Microsoft Windows, Mac OS X y Ubuntu.

Este simulador ofrece a los usuarios un subconjunto del total de las instrucciones que ARM UAL dispone. Este subconjunto está formado por las principales instrucciones

aritméticas, lógicas y saltos, entre otras. Además, se pueden destacar algunas características de este simulador, como pueden ser:

- Permite a los usuarios ver el valor que tenían anteriormente los registros porque dispone de un historial. Esta característica permite a los usuarios depurar de una manera más sencilla el código.
- Proporciona un cuadro de información del puntero que ofrece una información extra a los usuarios para entender el funcionamiento de los punteros de ARM.
- VisUAL ofrece la posibilidad de poder ver las operaciones que se realizan en el programa mediante una animación que muestra cómo se producen los cambios en los valores dependiendo de la operación realizada.
- Permite visualizar los accesos a memoria mostrándose la dirección base, el desplazamiento y los valores que han sido modificados. Además, la memoria se puede alinear por palabras o por bytes.
- Ofrece la posibilidad de visualizar la pila y mostrar la información sobre el comportamiento de esta y el puntero de pila del principio y del final.
- Indica la línea de código de destino de un salto mediante una flecha. El color de esta flecha será diferente si este salto es tomado o no.
- Permite detectar errores en tiempo de compilación y de ejecución. En ambos casos se muestra un mensaje detallando el error producido con la mayor precisión posible.
- VisUAL detecta posibles bucles infinitos en el código y se lo notifica al usuario.

Por último, se va a mostrar una figura con la interfaz de usuario de VisUAL.

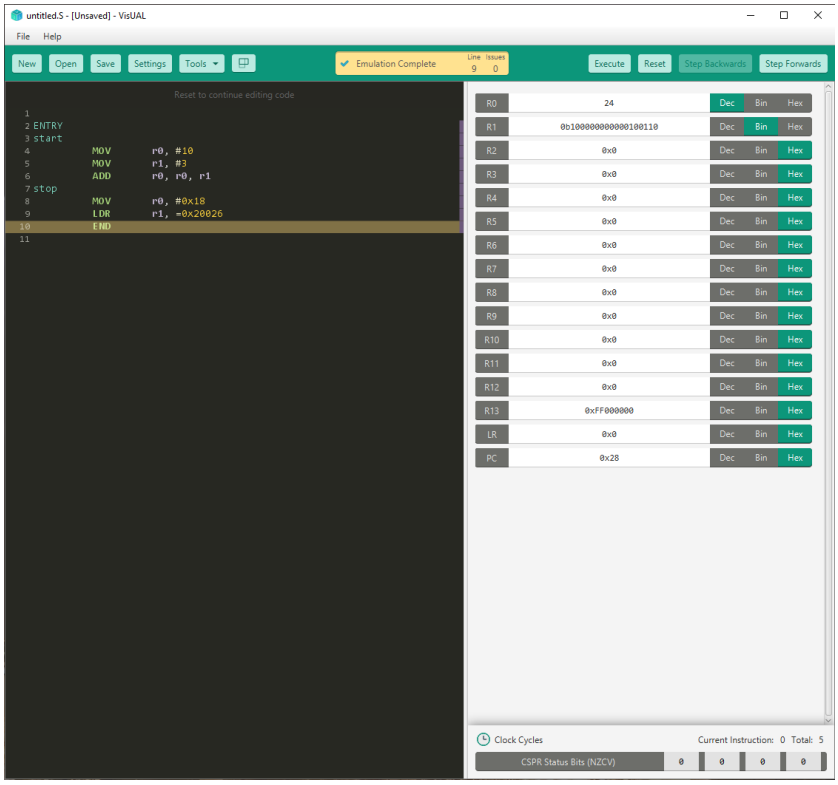


Fig. 2.5. Interfaz de VisUAL

2.1.5. ARMSim

ARMSim [6] es un simulador del lenguaje ensamblador ARM Thumb que fue desarrollado por los profesores de la asignatura “Estructura de Computadores” de la Universidad Jaume I de Castellón. Esta herramienta fue diseñada para que los estudiantes tuvieran una herramienta en la que complementar los conocimientos vistos en clase y, es por ello, por lo que este simulador es multiplataforma y gratuito.

Este simulador tiene una característica principal y es que el motor de simulación y la interfaz de usuario están desacoplados. Esta decisión fue tomada por sus diseñadores para permitir el uso del simulador en distintas interfaces gráficas o incluso de forma remota. El motor de simulación se denomina ARMSim y la interfaz de usuario QtARMSim. A continuación, se muestra una figura en la que se puede observar la interacción de los distintos componentes.

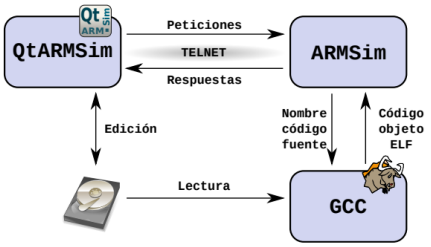


Fig. 2.6. Interacción de ARMSim y QtARMSim

En primer lugar, se van a exponer las características principales del componente ARMSim:

- ARMSim no implementa un ensamblador propio, en su lugar, utiliza un ensamblador ARM como puede ser GCC de GNU y, por lo tanto, ARMSim se encarga de interpretar el código que este genera.
- Este componente no ofrece interfaz de usuario. En su lugar ofrece un puerto donde escucha los comandos introducidos.
- ARMSim permite ensamblar un código ensamblador, consultar y modificar los registros y la memoria, consultar y definir puntos de ruptura, ejecutar el código desde una posición determinada y ejecutar el programa paso a paso.

Una vez descritas las características de ARMSim se van a exponer las características principales de QtARMSim:

- QtARMSim ofrece un editor de código ensamblador donde se escribirá el programa a ejecutar. Además, este editor es capaz de reconocer el lenguaje ensamblador ARM Thumb que permite colorear el código para una mejor comprensión.
- Permite observar toda la información relativa a las instrucciones cuando se va a ejecutar un programa. Esta información es: dirección de memoria de la instrucción, instrucción escrita en hexadecimal, instrucción expresada en ensamblador e instrucción original, es decir, la que ha introducido el usuario.
- Los paneles de la interfaz de usuario son empotrables, que quiere decir que los usuarios pueden cerrarlos, reubicarlos o mostrarlos como ventanas flotantes pudiendo siempre volver a la interfaz por defecto.

- QtARMSim ofrece la posibilidad de ejecutar un programa paso a paso, todo a la vez o hasta un punto de ruptura.

Por último, se muestra una imagen de la interfaz de usuario de QtARMSim.

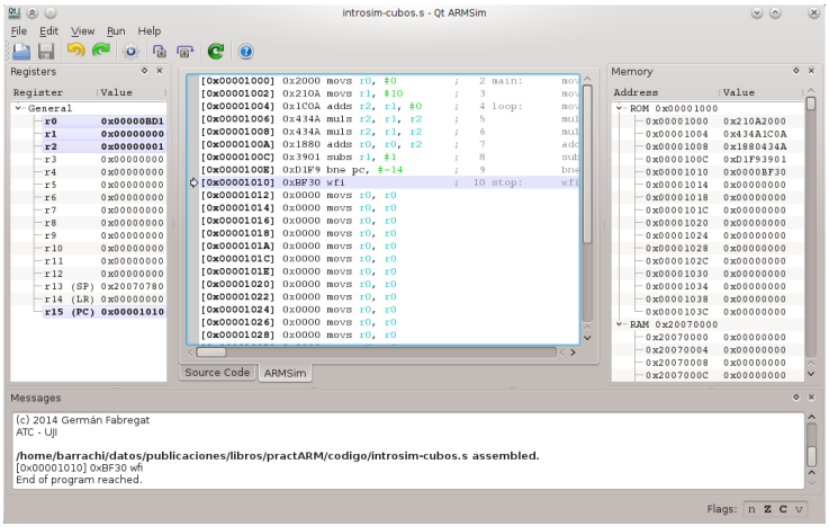


Fig. 2.7. Interfaz de QtARMSim

2.1.6. Comparativa de los simuladores

Actualmente existen diversos simuladores que se pueden utilizar como herramienta de apoyo en las asignaturas de Estructura y Arquitectura de Computadores. Por ello, a continuación, se van a comparar los distintos simuladores analizados anteriormente junto con el simulador que se va a desarrollar sobre una serie de características. Estas características son:

- Multiplataforma: el simulador se puede utilizar en diferentes plataformas como pueden ser ordenadores o dispositivos móviles, además de distintos Sistemas Operativos.
- Interactivo: ofrece una interfaz de usuario interactiva que facilita al usuario la interacción con el simulador.
- Arquitectura editable: la arquitectura cargada en el simulador se puede editar para crear nuevas arquitecturas.

- Genérico: el simulador es capaz de simular el funcionamiento de distintas arquitecturas.
- Ejecución de un binario: el simulador es capaz de ejecutar un código ensamblador mediante un binario.

TABLA 2.1. COMPARATIVA DE SIMULADORES

Simulador	SPIM	QtSPIM	MARS	WebMIPS	VisUAL	ARMSim	QtARMSim	CREATOR
Multiplataforma	✓	✓	-	✓	✓	✓	✓	✓
Interactivo	-	✓	✓	✓	✓	-	✓	✓
Arquitectura editable	-	-	-	-	-	-	-	✓
Genérico	-	-	-	-	-	-	-	✓
Ejecución de un binario	-	-	-	-	-	-	-	✓

Como se puede observar en la tabla anterior los simuladores analizados en el Estado del arte (Capítulo 2) tienen unas características similares, pero difieren en varias de ellas con respecto el simulador que se va a desarrollar, ya que este nuevo simulador pretende ser un simulador más completo que los ya existentes.

2.2. Tecnologías web

Debido a la multitud de dispositivos diferentes que existen actualmente la manera más habitual de ofrecer los contenidos digitales es mediante tecnologías web que permiten que estos recursos puedan ser utilizados en la gran mayoría de los dispositivos, independientemente de las características que cada uno tenga.

2.2.1. HTML5

Cuando se desarrolla una página web las tres tecnologías base que se utilizan en la mayoría de los casos son: Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) y JavaScript. Estas tecnologías aportan estructura, estilo y funcionalidad, respectivamente, a una página web. Todas estas tecnologías son dependientes entre sí y, por ello trabajan de forma conjunta en HTML5. Además, están estandarizadas por el World Wide Web Consortium (W3C), que está formado por cientos de empresas y organizaciones

entre las que están incluidas Microsoft, Apple y Google que son algunas de las empresas líderes en este sector [7]. A continuación, se va a detallar de una manera más exhaustiva cada una de estas tecnologías:

- **HTML**: Es un lenguaje que utiliza etiquetas para crear la estructura de una página web pero sin llegar a dotarle de funcionalidad.

  Este lenguaje tiene una característica muy importante y es que utiliza estas etiquetas y sus atributos, que proporcionan una información extra a la etiqueta, para indicar al navegador web el contenido que tiene que mostrar en la pantalla, es decir, las páginas web desarrolladas en HTML solo contienen texto y referencias externas en el caso de que el contenido sea multimedia o hipertexto siendo, el navegador web, el encargado de enlazar todos estos elementos y renderizar la página final.

  El lenguaje HTML ha tenido varias versiones a lo largo de su vida en las que se han añadido y eliminado diferentes características para adaptarlo a las nuevas plataformas y navegadores web. Actualmente, se está utilizando la versión 5 que ofrece más de 100 etiquetas diferentes y que ha mejorado en algunos aspectos a la versión 4 como es el soporte para audio y vídeo que en la actualidad es una parte esencial de las páginas web [7].

- **CSS**: Es un lenguaje que se usa para definir la apariencia de la página web, por esto se usa de forma conjunta con el lenguaje HTML que se encarga de definir el contenido de la página web.

  CSS permite indicar al navegador web diversas características como pueden ser: colores, familias de fuentes, tamaños, márgenes, entre otras. También CSS permite crear páginas web visualmente atractivas o *responsives*, consiguiendo adaptar en el dispositivo que se visualice el contenido de forma óptima.

  Lo que se pretende con este lenguaje es separar la definición y la presentación del contenido. Esto supone varias ventajas, como poder utilizar el mismo estilo en varios ficheros HTML, como ocurre en el caso de los navegadores web basados en habla (que son navegadores que leen en voz alta el contenido para las personas que sufren una discapacidad visual) puedan ignorar el código CSS porque no es necesario en este tipo de navegador [7].

- **JavaScript**: Es un lenguaje de programación ligero e interpretado que se creó para dotar a las páginas web de funcionalidad, aunque también es usado en algunos entornos sin navegador. Su sintaxis es similar a la de Java y a la de C++ para intentar reducir el esfuerzo de aprendizaje de este lenguaje.

  Es un lenguaje de script, esto quiere decir que este código se puede insertar dentro del código HTML o en un archivo externo y el navegador web será el encargado de ejecutar estas líneas en el ordenador del usuario sin tener que hacer uso de un servidor.

  JavaScript puede realizar multitud de tareas para proporcionar a las páginas web una funcionalidad y una personalización que con HTML no se consigue, en especial, en el área de la interfaz de usuario, creando una experiencia más gratificante a los usuarios.

  Actualmente, JavaScript está estandarizado en el ECMAScript, que es el estándar que utilizan los navegadores web modernos, y por ello, JavaScript es muy usado para aplicaciones web [8] [9].

2.2.2. Frameworks

Las aplicaciones web, hoy en día, son sistemas complejos que ofrecen interfaces de usuario que se asemejan a las aplicaciones de escritorio y que están diseñadas para que puedan ser utilizadas en diferentes plataformas. Estas aplicaciones también ofrecen complejas funcionalidades que añaden dificultad en la implementación, por esto, se usan para el desarrollo de las aplicaciones web *frameworks* que permiten acelerar el desarrollo, reutilizar código y promover buenas prácticas [10]. Algunos de los *frameworks* más usados actualmente son:

- **BootStrap**: es un *framework* para el diseño del *front-end* de las páginas web que fue desarrollado por Twitter [11]. Actualmente es de código abierto, lo que permite que sea usado en multitud de páginas web.

  El objetivo principal de Bootstrap es proporcionar una interfaz para el desarrollo de páginas web *responsive*. Además, también incluye plantillas para el diseño de botones, formularios y menús, entre otros. Estos elementos de diseño están desa-

rrollados mediante HTML y CSS que permite que sea compatible entre diferentes navegadores [12].

- **jQuery.js**: Es una biblioteca pequeña y rápida de código abierto de JavaScript que permite simplificar las interacciones que JavaScript realiza con la DOM.

  El objetivo principal de jQuery es simplificar la modificación del código HTML de forma dinámica y añadir soporte para animaciones y efectos, haciendo más atractiva la página web. Una de las características más importantes de jQuery es que soporta Ajax, lo que permite acceder a un servidor sin la necesidad de actualizar la página y que posibilita que la apariencia de estas páginas web sea muy similar a las de las aplicaciones de escritorio [13] [14].

- **React**: Es una biblioteca de JavaScript de código abierto que se utiliza para construir interfaces de usuario interactivas. Se encarga de actualizar la vista de la aplicación de forma automática cuando algún dato de esta cambia y que simplifica mucho el desarrollo y el código de la aplicación.

  React está basado en componentes, esto permite que cada componente sea capaz de manejar su propio estado sin tener en cuenta al resto de componentes que conforman la aplicación [15].

- **Angular**: Es un *framework* de JavaScript que utiliza la arquitectura Modelo-Vista-Controlador (MVC), actualmente es de código abierto y es mantenido por Google.

  Este *framework* se desarrolló para disminuir la distancia que separa HTML y JavaScript a la hora de diseñar el *Front End* de una página web, así como extender el desarrollo de páginas web que tienen una sola página y ampliar la funcionalidad que ofrece HTML.

  Para realizar todas estas funciones se utilizan una serie de atributos especiales que asocian los elementos HTML con la lógica de JavaScript. Esta capacidad permite manipular la DOM de una manera más sencilla mediante plantillas y el enlace de datos bidireccional que ofrece Angular y que permite sincronizar el modelo y la vista [16].

- **Vue.js**: Es un *framework* progresivo de código abierto que se utiliza para crear interfaces de usuario.

Vue.js se diseñó utilizando algunos principios de diseño que usan el lado servidor de las páginas web para aplicarlos a los elementos del HTML, facilitando, en gran medida, el desarrollo de las aplicaciones web. Además, este *framework* es muy útil cuando la página web es de una sola página ya que el navegador solo tiene que realizar el trabajo de cargar la página una vez y el resto de las interacciones que se realizan se producen en un segundo plano.

Por último, una de las principales características de Vue.js es el sistema de componentes que ofrece porque permite crear aplicaciones complejas formadas por componentes pequeños, independientes y reutilizables [17] [18].

- **Bootstrap + Vue**: Es un *framework* que proporciona de forma conjunta las características de Bootstrap y de Vue.js explicadas anteriormente [19].

2.3. Entornos de desarrollo de aplicaciones móviles

Actualmente, los dispositivos móviles son muy usados y poseen la ventaja, frente a los ordenadores, de que se pueden transportar de una manera sencilla y así poderlos usar en cualquier momento y lugar. Por esto han surgido diferentes entornos de desarrollo de aplicaciones móviles que nos permiten crear aplicaciones para cada una de las plataformas móviles. A continuación, se van a explicar Apache Cordova e Ionic que son dos de los entornos más usados actualmente.

2.3.1. Apache Cordova

Apache Cordova [20] es un entorno de desarrollo de aplicaciones móviles muy popular y de código abierto. Este entorno de desarrollo permite crear aplicaciones móviles híbridas utilizando tecnologías web habituales como HTML, CSS y JavaScript. Además de usar estas tecnologías, Apache Cordova, ofrece diferentes APIs para otorgar a la aplicación distintas capacidades que son elementales para el uso de una aplicación en un dispositivo móvil. Las funcionalidades más comunes son:

- Procesar listas de contactos del dispositivo.

- Procesar archivos del dispositivo.
- Utilizar la cámara del dispositivo.
- Acceder a la galería de imágenes del dispositivo.
- Acceder a la ubicación del dispositivo.

Apache Cordova permite crear aplicaciones para distintas plataformas y Sistemas Operativos, como pueden ser: Android, iOS, Windows Phone, Ubuntu o Web. Esto se puede realizar porque Apache Cordova crea aplicaciones híbridas, lo que quiere decir que utiliza tecnologías web comunes, mientras que las aplicaciones nativas se desarrollan utilizando un lenguaje de programación específico para cada plataforma. En el caso de Android se utiliza Java, para iOS se emplea Objective-C y para Windows Phone se usa .NET. Que las aplicaciones desarrolladas por este entorno de desarrollo sean híbridas es una gran ventaja ya que con el mismo código se puede usar en varias plataformas distintas, sin tener que desarrollar una aplicación con un código diferente para cada plataforma como sucedería con entornos de desarrollo de aplicaciones nativas, disminuyendo considerablemente el tiempo de desarrollo de una aplicación. Sin embargo, las aplicaciones creadas de forma nativa tienen mejor rendimiento que las aplicaciones híbridas, ya que el código usado para implementar estas está más optimizado.

Los componentes principales de una aplicación creada con Apache Cordova son los archivos HTML, CSS, JavaScript y en algunas ocasiones archivos de ayuda, como pueden ser archivos JSON, que son necesarios para el correcto funcionamiento de la aplicación. En el caso de que la aplicación necesite acceder a características nativas del dispositivo como puede ser el uso de la cámara utilizará la API que proporciona Apache Cordova para estos casos y que se encarga de traducir las llamadas a la API en JavaScript a llamadas a la API del dispositivo nativo, para ello hace uso de una capa puente. A continuación, se muestra una figura de cómo es la arquitectura de aplicación desarrollada en Apache Cordova.

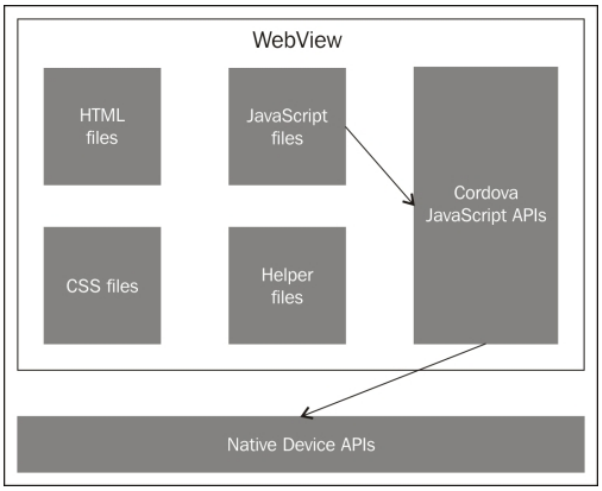


Fig. 2.8. Arquitectura de una aplicación creada con Apache Cordova

En la ilustración anterior se puede observar que existe un componente llamado *WebView* que es un navegador que se encarga de cargar el contenido web local con el que esta creada la aplicación. En este caso las aplicaciones suelen estar formadas por un único fichero HTML que ocupa todo el espacio disponible de la pantalla del dispositivo.

2.3.2. Ionic

Ionic [21] es un entorno de desarrollo de aplicaciones móviles híbridas de código abierto que, como se explicó en la sección anterior (Sección 2.3.1 Apache Cordova), son aplicaciones que se desarrollan utilizando las tecnologías web comunes (HTML, CSS y JavaScript).

Ionic está desarrollado con Angular y con Apache Cordova por lo que permite utilizar la API de Angular para crear interfaces de usuario más atractivas para los usuarios y, además, al usar Apache Cordova pueden utilizar todas las funcionalidades que este *framework* de desarrollo ofrece, como puede ser la API de acceso a las funciones nativas del dispositivo móvil.

Esta herramienta permite crear aplicaciones multiplataforma en las que el estilo de la aplicación creada es similar al de las aplicaciones nativas, puesto que dispone de hojas de estilo del aspecto nativo de Android e iOS. Esto es una ventaja ya que cada plataforma tiene definidos unos estilos que permiten diferenciarla de otras plataformas y que se deben seguir siempre que sea posible.

Capítulo 3

Análisis

En este capítulo se va a detallar el sistema a desarrollar mediante los requisitos que lo definen. Estos requisitos aportan la información necesaria para continuar con el desarrollo del simulador en los capítulos siguientes así como un análisis profundo de su funcionamiento.

Para la obtención de los requisitos el tutor ha realizado la labor de cliente y el estudiante el rol de analista.

En la Sección 3.1 se realiza una breve descripción del proyecto que se desarrolla. Por otro lado, en la Sección 3.2, se realiza la especificación de los requisitos de usuario y de los requisitos software, además, también se definen diferentes casos de uso de la herramienta. Por último, este capítulo trata en la Sección 3.3 el marco regulador, que detalla las regulaciones y leyes que afectan al software.

3.1. Descripción del proyecto

Este proyecto tiene como objetivo crear un simulador genérico para programar en lenguaje ensamblador y que muestre con realismo el funcionamiento a bajo nivel de un computador, además, de permitir a los estudiantes la posibilidad de simular diferentes arquitecturas sobre una misma herramienta.

Actualmente, los simuladores que existen para programar en ensamblador están enfocados a simular una única arquitectura, como puede ser MIPS o ARM, pero no pa-

ra simular diferentes arquitecturas sobre una misma herramienta, lo que perjudica a los alumnos ya que tienen que aprender el funcionamiento de cada una de las herramientas disminuyendo el tiempo del que disponen para aprender el funcionamiento de cada una de las arquitecturas. Asimismo, los simuladores educativos deben simular el funcionamiento de un computador con la mayor precisión posible, pero siempre siendo intuitivos y fáciles de usar para no generar problemas a los estudiantes que lo usan.

Por ello se propone un nuevo simulador que sea capaz de ejecutar programas ensamblador sobre diferentes tipos de arquitecturas. Este simulador, además, permitirá a los alumnos definir los juegos de instrucciones, la arquitectura y los componentes en los que se ejecutará el código ensamblador desarrollado. Esto posibilitará a los estudiantes comprender mejor los conocimientos vistos en las clases teóricas, así como realizar las prácticas de la asignatura en la herramienta.

3.2. Requisitos

En esta sección se van a explicar los diferentes requisitos que tiene el sistema. Para la especificación de los requisitos se han utilizado las prácticas definidas por el IEEE [22]. Estas prácticas indican que una buena especificación de requisitos debe explicar la funcionalidad del software, las interfaces externas, el rendimiento del sistema y las restricciones del diseño. Además, la especificación de requisitos debe ser:

- **Correcta**: todos los requisitos definidos tienen que reflejar una necesidad real en el sistema.
- **No ambigua**: cada requisito debe tener una única interpretación.
- **Completa**: todos los requisitos destacados están incluidos en la especificación.
- **Consistente**: los requisitos no son contradictorios entre sí.
- **Clasificada**: los requisitos son clasificados por importancia y por su estabilidad.
- **Verificable**: los requisitos tienen que ser verificables mediante un proceso finito y no costoso.

- **Modificable**: la especificación de requisitos debe poder ser modificada de forma sencilla, completa y consistente.
- **Trazable**: los requisitos deben ser trazables mediante su origen y se puede hacer de manera sencilla referencia en otras etapas.

Para la especificación de los requisitos se parte de los requisitos de usuario, que son condiciones informales que el cliente determina y que definen el funcionamiento del sistema. A partir de estos requisitos se crearán los requisitos de software, que se encargarán de dirigir el diseño, proporcionando una información más concisa sobre el funcionamiento y las diferentes características del sistema. Los requisitos se estructuran según el siguiente esquema:

1. **Requisitos de usuario**
   - *a*) **Capacidad**: representa lo que necesitan los usuarios para resolver un problema o lograr un objetivo.
   - *b*) **Restricción**: son las restricciones que se imponen por los usuarios sobre cómo se debe resolver el problema o como se debe alcanzar el objetivo.
2. **Requisitos de software**
   - *a*) **Funcionales**:
     1) **Funcional**: especifica lo que tiene que hacer el sistema.
     2) **Inverso**: indica las restricciones que el sistema tiene que cumplir.
   - *b*) **No funcionales**:
     1) **Rendimiento**: expone los requisitos mínimos de rendimiento que el sistema debe cumplir.
     2) **Interfaz**: requisitos que están vinculados con la interfaz de usuario del sistema.
     3) **Escalabilidad**: indica la capacidad de adaptación que tiene el sistema para cargas de trabajo cada vez mayores.
     4) **Plataforma**: concreta las plataformas en las que funcionará el sistema.

Una vez definido el esquema de clasificación de los requisitos se va a mostrar la plantilla que se utilizará para la especificación de requisitos de usuario y de software.

TABLA 3.1. PLANTILLA DE LA ESPECIFICACIÓN DE REQUISITOS

Identificador	Identificador del requisito.
Nombre	Nombre del requisito.
Tipo	Muestra la categoría en la que se clasifica el requisito según el esquema anterior.
Origen	Procedencia del requisito, pudiendo ser el usuario u otro requisito.
Prioridad	Indica la prioridad del requisito para ser cumplido. Un requisito puede ser: *esencial*, *deseable* u *opcional*.
Estabilidad	Indica si el requisito puede ser modificado a lo largo del desarrollo. Puede clasificarse como: *estable* o *inestable*.
Descripción	Explicación exhaustiva del requisito.

En el caso de los requisitos de usuario el identificador tendrá la siguiente estructura: RU-XYY, donde la X indica el tipo de requisito de usuario que podrá ser: requisito de capacidad (C) o requisito de restricción (R). YY corresponde al número del requisito dentro de su subcategoría.

Para los requisitos software el formato de identificación será: RW-X-YZZ, indicando la X si se trata de un requisito funcional (F) o de un requisito no funcional (NF), la Y señala la subcategoría del requisito, que puede ser: funcional (F), inverso (I), rendimiento (P), interfaz (UI), escalabilidad (S) o plataforma (PL) y, por último, ZZ indica el número del requisito dentro de su subcategoría.

3.2.1. Requisitos de usuario

En este apartado se van a indicar los diferentes requisitos de usuario.

Requisitos de capacidad

A continuación, se muestran los requisitos de capacidad que se han definido.

TABLA 3.2. REQUISITO DE USUARIO RU-C01

Identificador	RU-C01
Nombre	Simulación de diferentes arquitecturas
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador será capaz de simular el comportamiento de diferentes arquitecturas.

TABLA 3.3. REQUISITO DE USUARIO RU-C02

Identificador	RU-C02
Nombre	Selección de la arquitectura a simular
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador ofrecerá la posibilidad de seleccionar la arquitectura a simular.

TABLA 3.4. REQUISITO DE USUARIO RU-C03

Identificador	RU-C03
Nombre	Arquitecturas ofrecidas
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Inestable
Descripción	La herramienta ofrecerá la definición de las arquitecturas MIPS y ARM.

TABLA 3.5. REQUISITO DE USUARIO RU-C04

Identificador	RU-C04
Nombre	Operaciones con las arquitecturas disponibles
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema permitirá añadir y borrar arquitecturas que están disponibles para realizar simulaciones.

TABLA 3.6. REQUISITO DE USUARIO RU-C05

Identificador	RU-C05
Nombre	Operaciones con la arquitectura cargada para la simulación
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador permitirá guardar y volver a la definición por defecto de la arquitectura cargada.

TABLA 3.7. REQUISITO DE USUARIO RU-C06

Identificador	RU-C06
Nombre	Definición de los componentes de la arquitectura
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema deberá permitir al usuario crear, editar y borrar registros.

TABLA 3.8. REQUISITO DE USUARIO RU-C07

Identificador	RU-C07
Nombre	Definición del juego de instrucciones de la arquitectura
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta deberá permitir al usuario crear, editar y borrar instrucciones y pseudoinstrucciones.

TABLA 3.9. REQUISITO DE USUARIO RU-C08

Identificador	RU-C08
Nombre	Definición de código ensamblador
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe permitir al usuario definir el código ensamblador que se va a simular siguiendo el formato de las instrucciones que ofrece la arquitectura cargada.

TABLA 3.10. REQUISITO DE USUARIO RU-C09

Identificador	RU-C09
Nombre	Operaciones con el código ensamblador
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema debe permitir al usuario cargar el código ensamblador desde un fichero y exportarlo a un fichero.

TABLA 3.11. REQUISITO DE USUARIO RU-C10

Identificador	RU-C10
Nombre	Compilado del código ensamblador
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe ser capaz de compilar el código ensamblador introducido por el usuario y, en caso de encontrar errores en el código, notificarlos.

TABLA 3.12. REQUISITO DE USUARIO RU-C11

Identificador	RU-C11
Nombre	Operaciones con el binario del código ensamblador
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema debe permitir al usuario cargar una biblioteca de funciones en binario desde un fichero y guardarlo en un fichero.

TABLA 3.13. REQUISITO DE USUARIO RU-C12

Identificador	RU-C12
Nombre	Diferentes tipos de simulación
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe ser capaz de simular el programa ensamblador instrucción a instrucción o el programa completo.

TABLA 3.14. REQUISITO DE USUARIO RU-C13

Identificador	RU-C13
Nombre	Definición de *breakpoints*
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta permitirá al usuario definir *breakpoints* en la ejecución de un código ensamblador.

TABLA 3.15. REQUISITO DE USUARIO RU-C14

Identificador	RU-C14
Nombre	Información que se muestra
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador mostrará al usuario el estado en el que se encuentra el simulador después de ejecutar una instrucción.

TABLA 3.16. REQUISITO DE USUARIO RU-C15

Identificador	RU-C15
Nombre	Modificación del valor de los registros
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema permitirá al usuario modificar el valor actual de un registro.

TABLA 3.17. REQUISITO DE USUARIO RU-C16

Identificador	RU-C16
Nombre	Reiniciar simulación
Tipo	Capacidad
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta permitirá al usuario reiniciar la simulación con los valores por defecto.

TABLA 3.18. REQUISITO DE USUARIO RU-C17

Identificador	RU-C17
Nombre	Registro de notificaciones
Tipo	Capacidad
Origen	Cliente
Prioridad	Opcional
Estabilidad	Estable
Descripción	El simulador tendrá un registro de todas las notificaciones mostradas al usuario.

TABLA 3.19. REQUISITO DE USUARIO RU-C18

Identificador	RU-C18
Nombre	Copia de seguridad
Tipo	Capacidad
Origen	Analista
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador, automáticamente, realizará copias de seguridad de la arquitectura actual y el código ensamblador y, si existe, una copia de seguridad. Se le notificará al usuario al iniciar la herramienta.

Requisitos de restricción

A continuación, se muestran los requisitos de restricción que se han definido.

TABLA 3.20. REQUISITO DE USUARIO RU-R01

Identificador	RU-R01
Nombre	Multiplataforma
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador se podrá utilizar en diferentes tipos de plataforma, como pueden ser ordenadores o dispositivos móviles.

TABLA 3.21. REQUISITO DE USUARIO RU-R02

Identificador	RU-R02
Nombre	Navegadores web
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema debe poder utilizarse en los siguiente navegadores web: Microsoft Edge 30+, Mozilla Firefox 45+, Google Chrome 50+ y Safari 10+.

TABLA 3.22. REQUISITO DE USUARIO RU-R03

Identificador	RU-R03
Nombre	Aplicación Android
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La aplicación para el Sistema Operativo Android debe poderse ejecutar en la versión 6 y superiores.

TABLA 3.23. REQUISITO DE USUARIO RU-R04

Identificador	RU-R04
Nombre	Interfaz de usuario
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario se tiene que poder mostrar tanto en ordenadores como en dispositivos móviles.

TABLA 3.24. REQUISITO DE USUARIO RU-R05

Identificador	RU-R05
Nombre	Tiempo de ejecución de instrucción
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo medio que se tarda en ejecutar una instrucción ensamblador no debe sobrepasar los 0.5 segundos.

TABLA 3.25. REQUISITO DE USUARIO RU-R06

Identificador	RU-R06
Nombre	Ejecución de forma local
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador se ejecutará en el dispositivo del usuario, por lo que el servidor solo es necesario para acceder a la herramienta.

TABLA 3.26. REQUISITO DE USUARIO RU-R07

Identificador	RU-R07
Nombre	Conexión a internet
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta no necesitará conexión a internet para funcionar.

TABLA 3.27. REQUISITO DE USUARIO RU-R08

Identificador	RU-R08
Nombre	Tecnología de desarrollo
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema se desarrollará mediante HTML5.

TABLA 3.28. REQUISITO DE USUARIO RU-R09

Identificador	RU-R09
Nombre	Tamaño máximo de la arquitectura
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador soportará arquitecturas de 64 bits como máximo.

TABLA 3.29. REQUISITO DE USUARIO RU-R10

Identificador	RU-R10
Nombre	Memoria alineada
Tipo	Restricción
Origen	Cliente
Prioridad	Esencial
Estabilidad	Estable
Descripción	Los datos almacenados en la memoria deben estar alineado.

3.2.2. Modelo de casos de uso

Para mostrar las diferentes acciones que los futuros usuarios podrán realizar en el simulador, se han creado diferentes casos de uso. A continuación, se van a mostrar tres diagramas que representan las acciones que los usuarios pueden realizar para gestionar la arquitectura (Figura 3.1), gestionar el código ensamblador (Figura 3.2) y realizar la simulación de un programa (Figura 3.3).

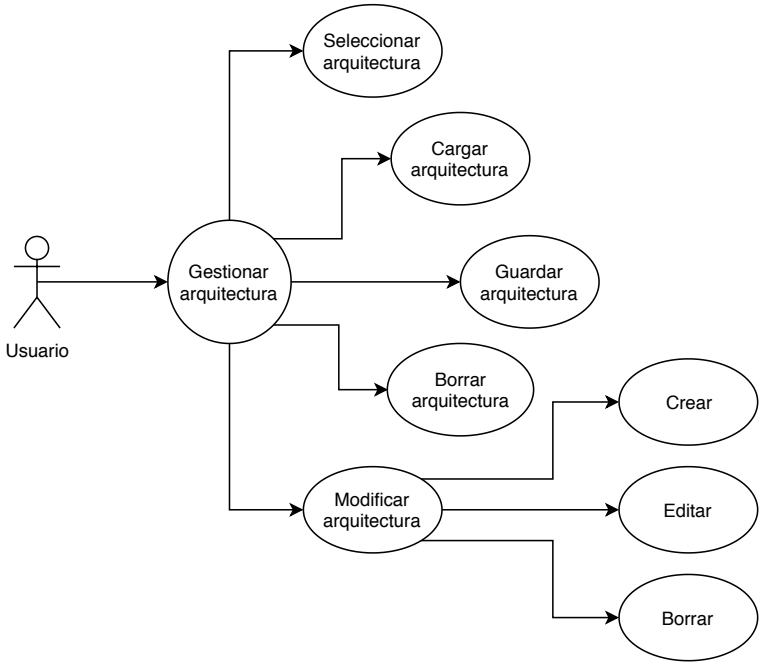


Fig. 3.1. Casos de uso de la gestión de la arquitectura

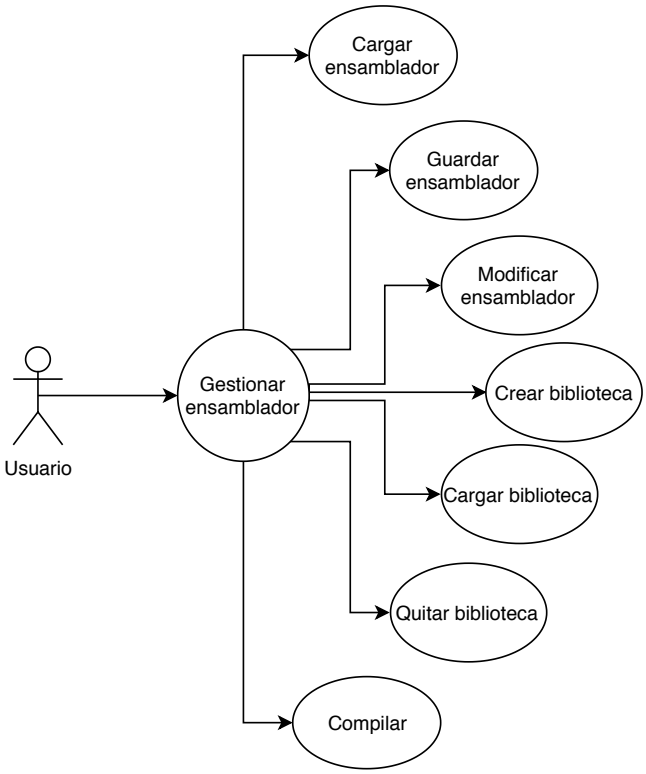


Fig. 3.2. Casos de uso de la gestión del código ensamblador

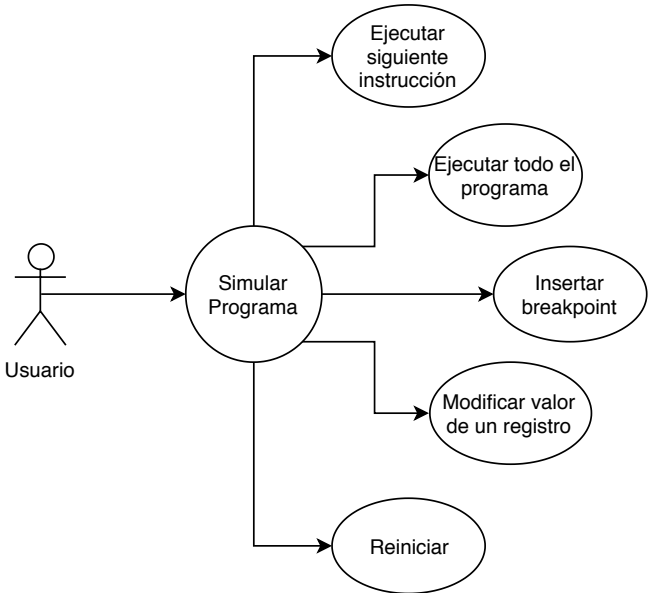


Fig. 3.3. Casos de uso de la simulación

Para una mejor explicación de cada uno de los casos de uso definidos en los diagramas anteriores, se va a hacer uso de la tabla siguiente.

TABLA 3.30. PLANTILLA PARA LOS CASOS DE USO

Identificador	Identificador del caso de uso.
Nombre	Nombre del caso de uso.
Actores	Actor o actores que están involucrados en la realización del caso de uso.
Objetivo	Indica el objetivo que tiene el caso de uso.
Precondiciones	Indica cuál tiene que ser el estado del sistema para que se pueda realizar el caso de uso.
Postcondiciones	Muestra cómo va a ser el estado del sistema cuando se realice el caso de uso.
Escenario	Describe los pasos que se tienen que producir para que se realice el caso de uso.

Una vez definida la plantilla que se va a usar para detallar los casos de uso, es necesario indicar el formato que va a seguir el identificador, este será: CU-XX, siendo XX el número del caso de uso.

A continuación, se van a presentar los casos de uso.

TABLA 3.31. CASO DE USO CU-01

Identificador	CU-01
Nombre	Seleccionar arquitectura.
Actores	Usuario
Objetivo	Cargar en el simulador una arquitectura predefinida.
Precondiciones	Ninguna.
Postcondiciones	La arquitectura se carga en el simulador
Escenario	▪ El usuario selecciona la tarjeta de la arquitectura que desea cargar.

TABLA 3.32. CASO DE USO CU-02

Identificador	CU-02
Nombre	Cargar arquitectura.
Actores	Usuario
Objetivo	Cargar una arquitectura en el simulador desde un fichero.
Precondiciones	Ninguna.
Postcondiciones	La arquitectura se carga en el simulador.
Escenario	▪ El usuario pulsa el botón “Load Architecture”. ▪ Indica un nombre para identificar a la arquitectura. ▪ Selecciona el fichero que contiene la definición de la arquitectura y acepta. ▪ Selecciona la tarjeta de la arquitectura que ha cargado.

TABLA 3.33. CASO DE USO CU-03

Identificador	CU-03
Nombre	Guardar arquitectura.
Actores	Usuario
Objetivo	Guardar la definición de la arquitectura que se está usando en el simulador en un fichero.
Precondiciones	El usuario tiene que tener una arquitectura cargada en el simulador.
Postcondiciones	Se genera un fichero que contiene la definición de la arquitectura que se estaba utilizando.
Escenario	▪ El usuario pulsa el botón “Architecture”. ▪ pulsa el botón “Save”. ▪ Indica, si desea, un nombre para el fichero y acepta.

TABLA 3.34. CASO DE USO CU-04

Identificador	CU-04
Nombre	Borrar arquitectura.
Actores	Usuario
Objetivo	Borra una arquitectura que ha sido cargada anteriormente por el usuario.
Precondiciones	Haber cargado anteriormente una nueva arquitectura en el simulador.
Postcondiciones	Se borra la arquitectura indicada.
Escenario	▪ El usuario pulsa el botón “Delete” de la arquitectura que desea borrar.

TABLA 3.35. CASO DE USO CU-05

Identificador	CU-05
Nombre	Modificar arquitectura - Crear.
Actores	Usuario
Objetivo	Crea un grupo de registros, un registro, una instrucción, una pseudoinstrucción o directiva.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se crea un nuevo elemento.
Escenario	▪ El usuario selecciona la pestaña del elemento que desea modificar. ▪ Pulsa el botón “New”. ▪ Completa el formulario y acepta.

TABLA 3.36. CASO DE USO CU-06

Identificador	CU-06
Nombre	Modificar arquitectura - Editar.
Actores	Usuario
Objetivo	Edita un grupo de registros, un registro, una instrucción, una pseudoinstrucción o directiva.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se edita el elemento seleccionado.
Escenario	▪ El usuario selecciona la pestaña del elemento que desea modificar. ▪ Pulsa el botón “Edit” del elemento que desea editar. ▪ Completa el formulario y acepta.

TABLA 3.37. CASO DE USO CU-07

Identificador	CU-07
Nombre	Modificar arquitectura - Borrar.
Actores	Usuario
Objetivo	Borra un grupo de registros, un registro, una instrucción, una pseudoinstrucción o directiva.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se borra el elemento seleccionado.
Escenario	▪ El usuario selecciona la pestaña del elemento que desea modificar. ▪ Pulsa el botón “Delete” del elemento que desea borrar. ▪ Acepta que desea realizar la operación.

TABLA 3.38. CASO DE USO CU-08

Identificador	CU-08
Nombre	Cargar código ensamblador.
Actores	Usuario
Objetivo	Carga en el simulador un código ensamblador desde un fichero.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	El código se carga en el simulador y, se puede visualizar este en el editor de texto del código ensamblador.
Escenario	▪ El usuario pulsa el botón “Assembly”. ▪ Pulsa el botón “Load”. ▪ Selecciona el fichero que contiene el código ensamblador y acepta.

TABLA 3.39. CASO DE USO CU-09

Identificador	CU-09
Nombre	Guardar código ensamblador.
Actores	Usuario
Objetivo	Guarda en un fichero el código ensamblador que está escrito en el simulador.
Precondiciones	Tener cargada una arquitectura en el simulador y, además, tener escrito un código ensamblador.
Postcondiciones	Se genera un fichero que contiene el código ensamblador que estaba escrito en el simulador.
Escenario	▪ El usuario pulsa el botón “Assembly”. ▪ Pulsa el botón “Save”. ▪ Indica, si desea, un nombre para el fichero y acepta.

TABLA 3.40. CASO DE USO CU-10

Identificador	CU-10
Nombre	Modificar código ensamblador.
Actores	Usuario
Objetivo	Edita el código ensamblador desde el simulador.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se muestran las modificaciones del usuario en el editor de texto.
Escenario	▪ El usuario pulsa el botón “Assembly”. ▪ Realiza las modificaciones que desea en el código.

TABLA 3.41. CASO DE USO CU-11

Identificador	CU-11
Nombre	Crear biblioteca.
Actores	Usuario
Objetivo	Crear una biblioteca que contiene funciones.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se genera un fichero que contiene el binario del código ensamblador que se había introducido en el simulador.
Escenario	■ El usuario pulsa el botón “Assembly”. ■ Escribe las funciones que se van a incluir en la biblioteca. ■ El usuario pulsa el botón “Create Library”. ■ Indica, si desea, un nombre para el fichero y acepta.

TABLA 3.42. CASO DE USO CU-12

Identificador	CU-12
Nombre	Cargar biblioteca.
Actores	Usuario
Objetivo	Cargar una biblioteca en el simulador.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	La biblioteca se almacena en el simulador y se puede consultar el nombre del las funciones que incluye.
Escenario	■ El usuario pulsa el botón “Assembly”. ■ El usuario pulsa el botón “Load Library”. ■ Selecciona el fichero que contiene la biblioteca y acepta.

TABLA 3.43. CASO DE USO CU-13

Identificador	CU-13
Nombre	Quitar biblioteca.
Actores	Usuario
Objetivo	Quita una biblioteca que está cargada en el simulador.
Precondiciones	Tener cargada una arquitectura y una biblioteca en el simulador .
Postcondiciones	La biblioteca se elimina del simulador y no puede ser usada.
Escenario	▪ El usuario pulsa el botón “Assembly”. ▪ El usuario pulsa el botón “Remove Library”.

TABLA 3.44. CASO DE USO CU-14

Identificador	CU-14
Nombre	Compilar el código ensamblador.
Actores	Usuario
Objetivo	Compila el código ensamblador para poder ejecutarlo.
Precondiciones	Tener cargada una arquitectura en el simulador, además de un código ensamblador.
Postcondiciones	Se compila el código ensamblador.
Escenario	▪ El usuario pulsa el botón “Assembly”. ▪ Pulsa el botón “Compile”.

TABLA 3.45. CASO DE USO CU-15

Identificador	CU-15
Nombre	Ejecutar siguiente instrucción.
Actores	Usuario
Objetivo	Ejecuta la siguiente instrucción.
Precondiciones	Tener compilado un programa ensamblador.
Postcondiciones	Ejecuta la instrucción en el simulador y avanza a la siguiente.
Escenario	▪ El usuario pulsa el botón “Simulator”. ▪ Pulsa el botón “Inst.”.

TABLA 3.46. CASO DE USO CU-16

Identificador	CU-16
Nombre	Ejecutar todo el programa.
Actores	Usuario
Objetivo	Ejecuta todo el programa ensamblador.
Precondiciones	Tener compilado un programa ensamblador.
Postcondiciones	Ejecuta el programa hasta que termina o hasta que encuentra un *breakpoint*.
Escenario	▪ El usuario pulsa el botón “Simulator”. ▪ Pulsa el botón “Run.”.

TABLA 3.47. CASO DE USO CU-17

Identificador	CU-17
Nombre	Insertar *breakpoint*.
Actores	Usuario
Objetivo	Inserta un *breakpoint* en una instrucción del código ensamblador que se desea simular.
Precondiciones	Tener compilado un programa ensamblador.
Postcondiciones	Se inserta un *breakpoint* en la simulación.
Escenario	▪ El usuario pulsa el botón “Simulator”. ▪ Pulsa sobre la fila en la que desea introducir el *breakpoint*.

TABLA 3.48. CASO DE USO CU-18

Identificador	CU-18
Nombre	Modificar el valor de un registro.
Actores	Usuario
Objetivo	Modifica el valor actual de un registro de la arquitectura cargada.
Precondiciones	Tener cargada una arquitectura en el simulador.
Postcondiciones	Se modifica el valor del registro indicado.
Escenario	▪ El usuario pulsa el botón “Simulator”. ▪ Pulsa sobre el registro que desea modificar. ▪ Introduce en decimal o en hexadecimal el nuevo valor del registro.

TABLA 3.49. CASO DE USO CU-19

Identificador	CU-19
Nombre	Reiniciar la simulación.
Actores	Usuario
Objetivo	Reinicia la simulación que se estaba realizando en el sistema.
Precondiciones	Tener compilado un programa ensamblador.
Postcondiciones	La simulación vuelve al inicio y el valor de los registros es modificado por el valor por defecto, al igual que el de la memoria.
Escenario	▪ El usuario pulsa el botón “Simulator”. ▪ Pulsa el botón “Reset”.

3.2.3. Requisitos software

En este apartado se van a indicar los diferentes requisitos software.

Requisitos funcionales

A continuación, se muestran los requisitos funcionales que se han definido.

TABLA 3.50. REQUISITO SOFTWARE RW-F-F01

Identificador	RW-F-F01
Nombre	Simulador genérico
Tipo	Funcional
Origen	RU-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador será capaz de simular el comportamiento de diferentes arquitecturas.

TABLA 3.51. REQUISITO SOFTWARE RW-F-F02

Identificador	RW-F-F02
Nombre	Elección de la arquitectura
Tipo	Funcional
Origen	RU-C02
Prioridad	Esencial
Estabilidad	Estable
Descripción	El usuario podrá elegir la arquitectura a simular.

TABLA 3.52. REQUISITO SOFTWARE RW-F-F03

Identificador	RW-F-F03
Nombre	Arquitectura MIPS32
Tipo	Funcional
Origen	RU-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta simulará el comportamiento de la arquitectura MIPS32.

TABLA 3.53. REQUISITO SOFTWARE RW-F-F04

Identificador	RW-F-F04
Nombre	Arquitectura ARM de 32 bits
Tipo	Funcional
Origen	RU-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta simulará el comportamiento de la arquitectura ARM de 32 bits.

TABLA 3.54. REQUISITO SOFTWARE RW-F-F05

Identificador	RW-F-F05
Nombre	Añadir nuevas arquitecturas
Tipo	Funcional
Origen	RU-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán añadir nuevas arquitecturas al simulador desde un fichero con formato .json.

TABLA 3.55. REQUISITO SOFTWARE RW-F-F06

Identificador	RW-F-F06
Nombre	Borrar arquitecturas disponibles
Tipo	Funcional
Origen	RU-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán borrar arquitecturas disponibles en el simulador.

TABLA 3.56. REQUISITO SOFTWARE RW-F-F07

Identificador	RW-F-F07
Nombre	Guardar arquitectura
Tipo	Funcional
Origen	RU-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá guardar la definición de una arquitectura en un fichero con formato .json.

TABLA 3.57. REQUISITO SOFTWARE RW-F-F08

Identificador	RW-F-F08
Nombre	Restablecer arquitectura
Tipo	Funcional
Origen	RU-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá volver a la definición por defecto de una arquitectura.

TABLA 3.58. REQUISITO SOFTWARE RW-F-F09

Identificador	RW-F-F09
Nombre	Crear registros
Tipo	Funcional
Origen	RU-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán crear nuevos registros.

TABLA 3.59. REQUISITO SOFTWARE RW-F-F10

Identificador	RW-F-F10
Nombre	Editar registros
Tipo	Funcional
Origen	RU-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán editar los atributos de los registros.

TABLA 3.60. REQUISITO SOFTWARE RW-F-F11

Identificador	RW-F-F11
Nombre	Borrar registros
Tipo	Funcional
Origen	RU-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán borrar los registros.

TABLA 3.61. REQUISITO SOFTWARE RW-F-F12

Identificador	RW-F-F12
Nombre	Crear instrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán crear nuevas instrucciones.

TABLA 3.62. REQUISITO SOFTWARE RW-F-F13

Identificador	RW-F-F13
Nombre	Editar instrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán editar los atributos de las instrucciones.

TABLA 3.63. REQUISITO SOFTWARE RW-F-F14

Identificador	RW-F-F14
Nombre	Borrar instrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán borrar las instrucciones.

TABLA 3.64. REQUISITO SOFTWARE RW-F-F15

Identificador	RW-F-F15
Nombre	Crear pseudoinstrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán crear nuevas pseudoinstrucciones.

TABLA 3.65. REQUISITO SOFTWARE RW-F-F16

Identificador	RW-F-F16
Nombre	Editar pseudoinstrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán editar los atributos de las pseudoinstrucciones.

TABLA 3.66. REQUISITO SOFTWARE RW-F-F17

Identificador	RW-F-F17
Nombre	Borrar pseudoinstrucciones
Tipo	Funcional
Origen	RU-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrán borrar las pseudoinstrucciones.

TABLA 3.67. REQUISITO SOFTWARE RW-F-F18

Identificador	RW-F-F18
Nombre	Definición de código ensamblador
Tipo	Funcional
Origen	RU-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	El usuario podrá definir el código ensamblador a simular.

TABLA 3.68. REQUISITO SOFTWARE RW-F-F19

Identificador	RW-F-F19
Nombre	Formato del código ensamblador
Tipo	Funcional
Origen	RU-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	El código ensamblador a simular debe tener el formato de las instrucciones de la arquitectura cargada.

TABLA 3.69. REQUISITO SOFTWARE RW-F-F20

Identificador	RW-F-F20
Nombre	Modificación del código ensamblador
Tipo	Funcional
Origen	RU-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	El código ensamblador podrá ser modificado por el usuario.

TABLA 3.70. REQUISITO SOFTWARE RW-F-F21

Identificador	RW-F-F21
Nombre	Carga del código ensamblador
Tipo	Funcional
Origen	RU-C09
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá cargar el código ensamblador desde un fichero con formato .s.

TABLA 3.71. REQUISITO SOFTWARE RW-F-F22

Identificador	RW-F-F22
Nombre	Exportar el código ensamblador
Tipo	Funcional
Origen	RU-C09
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá exportar el código ensamblador a un fichero con formato .s.

TABLA 3.72. REQUISITO SOFTWARE RW-F-F23

Identificador	RW-F-F23
Nombre	Compilación del código ensamblador
Tipo	Funcional
Origen	RU-C10
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder compilar el código ensamblador.

TABLA 3.73. REQUISITO SOFTWARE RW-F-F24

Identificador	RW-F-F24
Nombre	Notificación errores compilación
Tipo	Funcional
Origen	RU-C10
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se notificarán los errores que tiene el código ensamblador.

TABLA 3.74. REQUISITO SOFTWARE RW-F-F25

Identificador	RW-F-F25
Nombre	Carga de una biblioteca
Tipo	Funcional
Origen	RU-C11
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá cargar una biblioteca de funciones en binario desde un fichero con formato .o.

TABLA 3.75. REQUISITO SOFTWARE RW-F-F26

Identificador	RW-F-F26
Nombre	Exportar una biblioteca
Tipo	Funcional
Origen	RU-C11
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá exportar una biblioteca de funciones en binario a un fichero con formato .o.

TABLA 3.76. REQUISITO SOFTWARE RW-F-F27

Identificador	RW-F-F27
Nombre	Simulación instrucción a instrucción
Tipo	Funcional
Origen	RU-C12
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá simular el código ensamblador instrucción a instrucción.

TABLA 3.77. REQUISITO SOFTWARE RW-F-F28

Identificador	RW-F-F28
Nombre	Simulación programa completo
Tipo	Funcional
Origen	RU-C12
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá simular el código ensamblador sin interrupciones.

TABLA 3.78. REQUISITO SOFTWARE RW-F-F29

Identificador	RW-F-F29
Nombre	Definición de *breakpoints*
Tipo	Funcional
Origen	RU-C13
Prioridad	Esencial
Estabilidad	Estable
Descripción	El usuario podrá definir *breakpoints* en la simulación del código ensamblador.

TABLA 3.79. REQUISITO SOFTWARE RW-F-F30

Identificador	RW-F-F30
Nombre	Información de los registros en la simulación
Tipo	Funcional
Origen	RU-C14
Prioridad	Esencial
Estabilidad	Estable
Descripción	Durante la ejecución se debe mostrar el estado de los registros del simulador.

TABLA 3.80. REQUISITO SOFTWARE RW-F-F31

Identificador	RW-F-F31
Nombre	Información de la memoria en la simulación
Tipo	Funcional
Origen	RU-C14
Prioridad	Esencial
Estabilidad	Estable
Descripción	Durante la ejecución se debe mostrar el estado de la memoria del simulador.

TABLA 3.81. REQUISITO SOFTWARE RW-F-F32

Identificador	RW-F-F32
Nombre	Información de la instrucción simulada
Tipo	Funcional
Origen	RU-C14
Prioridad	Esencial
Estabilidad	Estable
Descripción	Durante la ejecución se debe mostrar la instrucción que se ha ejecutado.

TABLA 3.82. REQUISITO SOFTWARE RW-F-F33

Identificador	RW-F-F33
Nombre	Información de la próxima instrucción que se va a simular
Tipo	Funcional
Origen	RU-C14
Prioridad	Esencial
Estabilidad	Estable
Descripción	Durante la ejecución se debe mostrar la próxima instrucción que se va a ejecutar.

TABLA 3.83. REQUISITO SOFTWARE RW-F-F34

Identificador	RW-F-F34
Nombre	Nuevo valor de un registro en decimal
Tipo	Funcional
Origen	RU-C15
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá introducir un nuevo valor a un registro con un número decimal.

TABLA 3.84. REQUISITO SOFTWARE RW-F-F35

Identificador	RW-F-F35
Nombre	Nuevo valor de un registro en hexadecimal
Tipo	Funcional
Origen	RU-C15
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá introducir un nuevo valor a un registro con un número hexadecimal.

TABLA 3.85. REQUISITO SOFTWARE RW-F-F36

Identificador	RW-F-F36
Nombre	Reinicio de la simulación
Tipo	Funcional
Origen	RU-C16
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se podrá reiniciar la simulación con los valores por defecto.

TABLA 3.86. REQUISITO SOFTWARE RW-F-F37

Identificador	RW-F-F37
Nombre	Registro de notificaciones
Tipo	Funcional
Origen	RU-C17
Prioridad	Opcional
Estabilidad	Estable
Descripción	La herramienta dispone de un registro de notificaciones mostradas.

TABLA 3.87. REQUISITO SOFTWARE RW-F-F38

Identificador	RW-F-F38
Nombre	Creación de la copia de seguridad de la arquitectura
Tipo	Funcional
Origen	RU-C18
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador guarda la arquitectura en la caché del navegador en formato JSON cuando se compila un programa.

TABLA 3.88. REQUISITO SOFTWARE RW-F-F39

Identificador	RW-F-F39
Nombre	Creación de la copia de seguridad del código ensamblador
Tipo	Funcional
Origen	RU-C18
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador guarda el código ensamblador en la caché del navegador en formato JSON cuando se compila un programa.

TABLA 3.89. REQUISITO SOFTWARE RW-F-F40

Identificador	RW-F-F40
Nombre	Cargado de la copia de seguridad
Tipo	Funcional
Origen	RU-C18
Prioridad	Esencial
Estabilidad	Estable
Descripción	Si existe una copia de seguridad al iniciar la herramienta se notifica al usuario.

TABLA 3.90. REQUISITO SOFTWARE RW-F-I01

Identificador	RW-F-I01
Nombre	Tamaño máximo de la arquitectura
Tipo	Inverso
Origen	RU-R09
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema soportará arquitecturas de máximo 64 bits.

TABLA 3.91. REQUISITO SOFTWARE RW-F-I02

Identificador	RW-F-I02
Nombre	Memoria alineada
Tipo	Inverso
Origen	RU-R10
Prioridad	Esencial
Estabilidad	Estable
Descripción	Los datos almacenados en la memoria del sistema deben estar alineados.

Requisitos no funcionales

A continuación, se muestran los requisitos no funcionales que se han definido.

TABLA 3.92. REQUISITO SOFTWARE RW-NF-PL01

Identificador	RW-NF-PL01
Nombre	Multiplataforma
Tipo	Plataforma
Origen	RU-R01
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador se diseñará como una aplicación web.

TABLA 3.93. REQUISITO SOFTWARE RW-NF-PL02

Identificador	RW-NF-PL02
Nombre	Disponible en Microsoft Edge
Tipo	Plataforma
Origen	RU-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder utilizar en Microsoft Edge 30+.

TABLA 3.94. REQUISITO SOFTWARE RW-NF-PL03

Identificador	RW-NF-PL03
Nombre	Disponible en Mozilla Firefox
Tipo	Plataforma
Origen	RU-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder utilizar en Mozilla Firefox 45+.

TABLA 3.95. REQUISITO SOFTWARE RW-NF-PL04

Identificador	RW-NF-PL04
Nombre	Disponible en Google Chrome
Tipo	Plataforma
Origen	RU-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder utilizar en Google Chrome 50+.

TABLA 3.96. REQUISITO SOFTWARE RW-NF-PL05

Identificador	RW-NF-PL05
Nombre	Disponible en Safari
Tipo	Plataforma
Origen	RU-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder utilizar en Safari 10+.

TABLA 3.97. REQUISITO SOFTWARE RW-NF-PL06

Identificador	RW-NF-PL06
Nombre	Disponible en Android
Tipo	Plataforma
Origen	RU-R03
Prioridad	Esencial
Estabilidad	Estable
Descripción	Se debe poder utilizar en Android 6 y superiores.

TABLA 3.98. REQUISITO SOFTWARE RW-NF-PL07

Identificador	RW-NF-PL07
Nombre	Ejecución de forma local
Tipo	Plataforma
Origen	RU-R06
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema se ejecuta en el dispositivo del usuario.

TABLA 3.99. REQUISITO SOFTWARE RW-NF-PL08

Identificador	RW-NF-PL08
Nombre	Conexión a internet
Tipo	Plataforma
Origen	RU-R07
Prioridad	Esencial
Estabilidad	Estable
Descripción	El sistema no necesitará conexión a internet para funcionar.

TABLA 3.100. REQUISITO SOFTWARE RW-NF-PL09

Identificador	RW-NF-PL09
Nombre	Tecnología de desarrollo
Tipo	Plataforma
Origen	RU-R08
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta se desarrollará mediante HTML5 (HTML, CSS y JavaScript).

TABLA 3.101. REQUISITO SOFTWARE RW-NF-UI01

Identificador	RW-NF-UI01
Nombre	Interfaz de usuario PC
Tipo	Interfaz de usuario
Origen	RU-R04
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario se debe poder mostrar en ordenadores.

TABLA 3.102. REQUISITO SOFTWARE RW-NF-UI02

Identificador	RW-NF-UI02
Nombre	Interfaz de usuario dispositivos móviles
Tipo	Interfaz de usuario
Origen	RU-R04
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario se debe poder mostrar en dispositivos móviles.

TABLA 3.103. REQUISITO SOFTWARE RW-NF-R01

Identificador	RW-NF-R01
Nombre	Tiempo de ejecución de instrucción
Tipo	Rendimiento
Origen	RU-R05
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo medio que tarda en ejecutarse una instrucción no debe ser mayor de 0.5 segundos.

3.2.4. Matriz de trazabilidad de requisitos

Una vez especificados todos los requisitos software se va a mostrar una matriz de trazabilidad donde se puede apreciar que todos los requisitos de usuario tienen asociado al menos un requisito software.

TABLA 3.104. MATRIZ DE TRAZABILIDAD DE REQUISITOS

Requisitos	RU-C01	RU-C02	RU-C03	RU-C04	RU-C05	RU-C06	RU-C07	RU-C08	RU-C09	RU-C10	RU-C11	RU-C12	RU-C13	RU-C14	RU-C15	RU-C16	RU-C17	RU-C18	RU-R01	RU-R02	RU-R03	RU-R04	RU-R05	RU-R06	RU-R07	RU-R08	RU-R09	RU-R10
RW-F-F01	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F02	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F03	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F04	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F05	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F06	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F07	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F08	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F09	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F10	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F11	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F12	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F13	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F14	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F15	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F16	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F17	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F18	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F19	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F20	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F21	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F22	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F23	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F24	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F25	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F26	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F27	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F28	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F29	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F30	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F31	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F32	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F33	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F35	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F36	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F37	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RW-F-F39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RW-F-F40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RW-F-I01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-
RW-F-I02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
RW-NF-PL01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
RW-NF-PL02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RW-NF-PL03	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RW-NF-PL04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RW-NF-PL05	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RW-NF-PL06	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
RW-NF-PL07	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL08	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
RW-NF-PL09	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
RW-NF-UI01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RW-NF-UI02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RW-NF-R01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-

3.3. Marco regulador

En este apartado se van a exponer las licencias bajo las que se distribuyen los distintos *frameworks* y programas software utilizados, así como, las restricciones legales que se pueden aplicar a este proyecto.

3.3.1. Licencias de uso del software

Para el desarrollo de este proyecto se utilizaron diferentes *frameworks*. En primer lugar, Bootstrap se distribuye bajo una licencia de código abierto, concretamente bajo la licencia MIT [23]. Otro de los *framework* que se utiliza es JQuery.js que se ofrece bajo la licencia MIT y la Licencia Pública General de GNU (GPL) [24]. Por último, Bootstrap + Vue se ofrece a través de la licencia MIT.

Además de los *frameworks* anteriores, también se utiliza un entorno de desarrollo de aplicaciones móviles llamado Apache Cordova y que se ofrece bajo la licencia Apache 2.0 [25] de software libre.

3.3.2. Aspectos legales

En la mayor parte de las aplicaciones web que existen actualmente, los usuarios tienen que registrase para poder hacer uso de estas. Estos datos, de carácter confidencial, son almacenados en bases de datos y, por lo tanto, hay que proteger esta información para que otros usuarios no tengan acceso. Una de las soluciones utilizadas para este fin consiste en cifrar esta información mediante métodos criptográficos. En España, esta condición está regulada por el Reglamento General de Protección de Datos (RGPD) [26] que se encarga de regular este aspecto en toda la Unión Europea. Además, también existe la Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDGDD) [27].

Sin embargo, la aplicación web desarrollada no hace uso de funcionalidades que traten datos confidenciales porque en esta aplicación el usuario ejecutará programas en lenguaje ensamblador de manera local.

Por último, este simulador va a estar disponible como software de código abierto. Para ello, este software se encuentra bajo la Licencia Pública General Menor de GNU (LGPL) [24] y, por lo tanto, cualquier persona puede redistribuir o modificar dicho código siempre y cuando cumplan con los términos de esta licencia.

Capítulo 4

Diseño

En este capítulo se va a detallar de manera exhaustiva el diseño que va a tener el simulador. En primer lugar, en la Sección 4.1, se realizará un estudio de las diferentes alternativas que existen para realizar este proyecto hasta llegar a una solución final que cumpla con los requisitos establecidos en la Sección 3.2.1. A continuación, en la Sección 4.2, se describirá la arquitectura del simulador, explicando en detalle cada uno de los componentes que la forman.

4.1. Estudio de la solución final

Para realizar el estudio de la solución final se han tenido que tener en cuenta los requisitos de usuario que se especificaron en la Sección 3.2.1, ya que estos requisitos nos indicarán las capacidades y las limitaciones del sistema.

En estos requisitos se indicaba que el sistema debía ser multiplataforma, y por lo tanto, se ha decidido que el simulador sea una aplicación web porque este tipo de aplicaciones pueden ser ejecutadas por la gran mayoría de los dispositivos que hoy en día existen, así como adaptar la interfaz de usuario a diferentes tamaños de pantalla, que es un aspecto muy importante porque actualmente es muy utilizado por un gran número de dispositivos móviles.

Las aplicaciones web pueden tener diferentes modelos arquitectónicos. Una alternativa es realizar una aplicación web que tenga un modelo cliente-servidor. En este tipo

de arquitectura las tareas se distribuyen entre el servidor, que se encarga de proporcionar recursos o servicios, y el cliente, que se ocupa de realizar peticiones al servidor para que este le facilite recursos o servicios. Este modelo tiene la ventaja de que divide las responsabilidades. El servidor se encargará de realizar los cómputos necesarios y la gestión de la información y el cliente se ocupa de ofrecer la interfaz de usuario y ejecutar acciones que necesitan poca potencia de cómputo, como puede ser la validación de formularios. Otra opción consiste en que la aplicación web solo esté formada por la parte cliente y esta se encargue de ejecutar la aplicación web al completo. En este modelo arquitectónico el servidor no es necesario salvo para acceder al código de la aplicación en el caso de que este no se encuentre en el cliente, lo que permite que la aplicación se pueda ejecutar sin acceso a internet. Estos dos modelos son independientes del sistema operativo en el que se utiliza la aplicación, ya que dependen exclusivamente del navegador web que se utiliza para la ejecución de la aplicación web.

Una vez presentados estos dos modelos se ha decidido utilizar el segundo modelo. El motivo es que el modelo cliente-servidor no cumple con varios requisitos de usuario, ya que estos indican que la aplicación se debe ejecutar de forma local en el dispositivo del usuario y no es necesario tener conexión a internet.

El siguiente paso es determinar la tecnología web que se va a usar. La tecnología web que se usará para implementar la aplicación es HTML5, que está formado por HTML, CSS y JavaScript, tal y como se indicó en los requisitos de usuario. Para facilitar la implementación de la aplicación, así como dotarla de mayor funcionalidad y estilo, se han considerado diferentes *frameworks*. Estos *frameworks* tienen que ser compatibles con los navegadores y versiones indicadas en los requisitos y, además, proporcionar diferentes funcionalidades como el diseño *responsive*, que es una característica muy importante, ya que se trata de una aplicación multiplataforma, o un manejo lo más sencillo y automático posible de la DOM mediante el Modelo-Vista-Controlador (MVC) o el Modelo-Vista-Modelo de vista (MVVM). A continuación, se muestra una tabla con los distintos *frameworks* estudiados y las funciones que proporcionan.

TABLA 4.1. COMPARATIVA DE FRAMEWORKS

Framework	BootStrap	jQuery.js	React	Angualar	BootsTrap + Vue
Multiplataforma	✓	✓	✓	✓	✓
Ejecución en lado cliente	✓	✓	✓	✓	✓
Compatible con los navegadores indicados	✓	✓	✓	✓	✓
Manipulación DOM	-	✓	✓	✓	✓

Después de comparar los diferentes *frameworks*, se ha decidido usar para el desarrollo de la aplicación web jQuery.js [13] y Bootstrap + Vue [19], siendo este último elegido porque es un *framework* que utiliza de forma conjunta Bootstrap [12] y Vue.js [17], Bootstrap que es uno de los *frameworks* más usados para el desarrollo de las interfaces de usuario de las aplicaciones web y Vue.js, que es un *framework* más joven que Angular [16] y React [15]. Actualmente, es uno de los *framework* más usados y esto es debido a que este *framework* busca simplificar el funcionamiento de Angular y agrupar las mejores funcionalidades que proporcionan Angular y React en un mismo *framework*. Además, permite un manejo de la DOM muy sencillo, puesto que está diseñado mediante el patrón MVVM. Esta virtud es muy importante en esta aplicación debido a la gran cantidad de datos que se muestran en la interfaz de usuario y que además cambian de valor con mucha frecuencia. jQuery.js se eligió porque permite capturar de una manera sencilla los eventos que ocurren en la interfaz de usuario, así como las funcionalidades que ofrece para manejar ficheros en formato JSON. Por lo tanto, Bootstrap, React y Angular fueron descartados, ya que con estos dos *frameworks* se pueden realizar la gran mayoría de las funcionalidades que estos ofrecen de una manera más sencilla, porque la curva de aprendizaje de los *frameworks* seleccionados es mucho menor.

El simulador tiene que ser capaz de guardar y cargar ficheros con la definición de la arquitectura que el usuario estaba usando o deseaba usar en él, por ello, tras analizar los formatos XML [28] y JSON [29], se decidió hacer uso de ficheros con formato JSON, puesto que en este formato se pueden utilizar matrices, es más rápido de leer y escribir, es más corto y, además, en JavaScript existen funciones que permiten generar un texto en formato JSON a partir de un objeto y también para realizar la operación inversa, es decir, obtener un objeto a partir de un texto en formato JSON, lo que facilitaba mucho estas tareas.

Por último, en los requisitos de usuario también se indicaba que la aplicación tenía que estar disponible como una aplicación para Android. Después de analizar diferentes entornos de desarrollo de aplicaciones Android como Android Studio [30] y Apache Cordova [31], se decidió usar Apache Cordova, porque este entorno de desarrollo permite convertir una aplicación web implementada mediante HTML, CSS y JavaScript, que son las tecnologías web que se utilizan para desarrollar esta aplicación web, en una aplicación para este Sistema Operativo.

4.1.1. Solución final

Para que los alumnos de las asignaturas de Estructura y Arquitectura de Computadores dispongan de una herramienta que los permita poner en práctica los conocimientos impartidos en las clases teóricas y así comprender mejor los conceptos, se sugiere el diseño y desarrollo de un simulador web que permita ejecutar programas en lenguaje ensamblador mostrando, con el mayor realismo posible, el funcionamiento de un procesador.

Esta herramienta se desarrollará como aplicación web, ya que este tipo de aplicaciones pueden ser ejecutadas en cualquier dispositivo independientemente del Sistema Operativo que utilice el dispositivo y que solo dependen del navegador web en el que se ejecuta la aplicación. Esto permite que la aplicación pueda ser utilizada en cualquier momento y lugar en un dispositivo porque no se requiere que la aplicación sea instalada en el dispositivo en el que se va a utilizar.

Este simulador será implementado mediante HTML5, que engloba HTML, CSS y JavaScript, lo que permite que esta aplicación se pueda utilizar en ordenadores, *tablets*, *smartphones* y otros tipos de dispositivos que permitan el uso de un navegador web como puede ser Google Chrome, Mozilla Firefox, Microsoft Edge o Safari. Además de HTML5, también se usarán dos *frameworks*, que serán: Bootstrap + Vue y jQuery.js.

Además de desarrollarse este simulador como aplicación web, también se creará una aplicación para el Sistema Operativo Android, ya que un gran número de los dispositivos que se utilizan actualmente son dispositivos móviles y la gran mayoría utilizan Android.

Por consiguiente, esta solución permitirá ejecutar, en diferentes dispositivos, un si-

mulador genérico para programar en ensamblador sin la necesidad de realizar instalaciones ni conexión a internet.

4.2. Arquitectura del simulador

El simulador que se va a desarrollar está formado por diferentes componentes que desempeñarán una función determinada en el sistema. Estos son:

- Editor de arquitectura: permite definir y modificar la arquitectura que se va a usar en la simulación.
- Compilador: se encarga de compilar el código ensamblador introducido para que pueda ser ejecutado.
- Kernel de simulación: este componente se encarga de ejecutar el código compilado sobre la arquitectura definida.
- Interfaz de usuario: este componente se encarga de mostrar en pantalla el estado actual del simulador, además de capturar las acciones que realiza el usuario sobre esta.

El editor de arquitectura permite definir y modificar las diferentes características que tiene la arquitectura en la que se va a simular el código ensamblador que introduce el usuario, algunas de las características son: registros, instrucciones o directivas. Estas características son imprescindibles para realizar una simulación que sea similar a la realidad, omitiéndose otras que provocarían que esta definición y simulación fuera demasiado compleja para los alumnos.

El compilador se encarga de compilar el código ensamblador que introduce el usuario para que posteriormente pueda ser simulado sobre la arquitectura definida. Además, en caso de que el código contenga errores se indicará mediante un mensaje el error producido para que el usuario pueda subsanarlo.

El siguiente componente es el kernel de simulación, cuya tarea es ejecutar el código ensamblador compilado sobre la arquitectura definida en el editor de arquitectura.

Por último, la interfaz de usuario se encarga de mostrar en pantalla toda la información del estado actual del simulador, así como ser el componente sobre el que interacciona el usuario para realizar acciones en los componentes anteriores.

A continuación, se muestra un diagrama en el que se pueden ver los diferentes componentes que conforman el simulador y las interacciones que existen entre ellos:

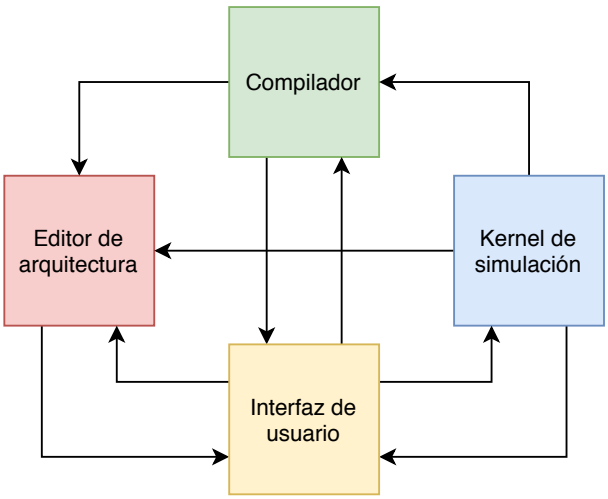


Fig. 4.1. Arquitectura de la aplicación

Como se puede observar la interfaz de usuario se comunica con el resto de los componentes, enviando y recibiendo información. Sin embargo, el compilador solo se comunica con el editor de arquitectura para obtener la información que necesita sobre la arquitectura para poder compilar el código correctamente y la interfaz de usuario. El kernel de simulación se comunica, con la interfaz de usuario, con el compilador para obtener las instrucciones que tiene que ejecutar y también con el editor de arquitectura para obtener la información que necesita para poder ejecutar el código, y por último, el editor de arquitectura que no interacciona con ningún otro componente para solicitar información, solo se encarga de proporcionarla.

4.2.1. Editor de arquitectura

El editor de arquitectura del simulador permite definir múltiples características de la arquitectura que se va a utilizar en la simulación, las características que se pueden definir son: conjuntos de registros, registros, instrucciones, pseudoinstrucciones y directivas. Este conjunto de características podrá ser editado, ya que se pueden crear nuevas

características, editarlas o borrarlas.

Cada una de las características anteriores estará constituida por diferentes atributos que serán los mínimos necesarios para poder realizar la simulación sin aumentar excesivamente la complejidad del sistema. A continuación, se va a detallar la definición de cada una de las características, así como su función.

Los conjuntos de registros se encargan de contener uno o varios registros, los cuales tienen unas características similares, como puede ser el tipo de registro. Los atributos de los conjuntos de registros son:

- Nombre (String): nombre único que identifica al conjunto de requisitos.
- Tipo (String): indica de que tipo son los registros que agrupa, este atributo podrá ser: registros de enteros, registros en coma flotante o registros de control.
- Doble precisión (Booleano): indica si los registros almacenados son de doble precisión o no, en el caso de que estos sean registros en coma flotante.

Los registros almacenan en su interior un valor numérico que podrá ser un número entero o decimal. Los atributos que definen a un registro son:

- Nombre (String): es un nombre único que identifica al registro.
- Número de bits (String): indica los bits que ocupa el registro.
- Valor (BigInt): indica el valor actual del registro.
- Valor por defecto (BigInt): especifica el valor que tiene el registro inicialmente y después de restablecer el sistema.
- Propiedades (Array de String): indica las propiedades de un registro, estas podrán ser: lectura y escritura.
- Registros de simple precisión (Array de String): este atributo será usado en el caso de que el registro sea de doble precisión, y se utiliza para indicar los dos registros de simple precisión a los que está asociado.

Las instrucciones se encargarán de realizar operaciones aritméticas, llamadas al sistema o accesos a memoria durante la simulación. Los atributos que tienen las instrucciones son:

- Nombre (String): nombre de la instrucción.
- Tipo (String): indica de que tipo es la instrucción definida.
- Código de operación (String): identificador único en código binario.
- Código de operación extendido (String): identificador único en binario que sustituye al código de operación en algunas instrucciones con operaciones aritméticas, ya que indica la operación que realiza la ALU.
- Número de palabras (Int): número de palabras que necesita la instrucción para ser definida y cargada en memoria.
- Campos: son las diferentes partes que forman la sintaxis de una instrucción, a su vez cada campo está formado por cuatro atributos, estos son:
  - Nombre (String): nombre del campo.
  - Tipo (String): indica el tipo de dato que contiene el campo, este podrá ser: código de operación, código de operación extendido, valor inmediato, dirección de memoria, registro entero, registro en coma flotante o registro de control.
  - Bit inicial (Int): define el bit de inicio del campo dentro de la instrucción.
  - Bit final (Int): indica el ultimo bit que ocupa este campo en la instrucción.
- Definición de sintaxis (String): define la sintaxis de la instrucción indicando la posición de los diferentes campos, así como los diferentes símbolos que contenga entre estos.
- Sintaxis con tipos (String): muestra la sintaxis de la instrucción indicando el tipo de dato que debe ir cada posición.
- Sintaxis con nombres (String): indica la sintaxis de la instrucción mostrando el nombre del campo que se sitúa en cada posición.

- Definición de la instrucción (String): define la funcionalidad de la instrucción.

A continuación, se muestra una imagen donde se puede observar los diferentes campos que forman una instrucción, junto con los bits de inicio y fin, el nombre del campo y el código de operación.

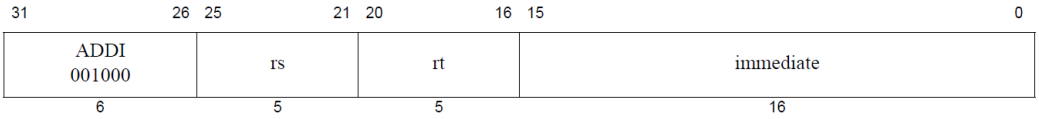


Fig. 4.2. Estructura de una instrucción

Las pseudoinstrucciones son instrucciones complejas que se dividen en varias instrucciones para realizar la acción deseada. Los atributos que la definen son muy similares a los de las instrucciones, pero tienen algunas variaciones y, por ello, se van a detallar a continuación.

- Nombre (String): nombre de la pseudoinstrucción.
- Número de palabras (Int): número de palabras que necesita la pseudoinstrucción para ser definida y cargada en memoria.
- Campos: son las diferentes partes que forman la sintaxis de una pseudoinstrucción, a su vez cada campo está formado por cuatro atributos que son:
  - Nombre (String): nombre del campo.
  - Tipo (String): indica el tipo de dato que contiene el campo, este podrá ser: código de operación, código de operación extendido, valor inmediato, dirección de memoria, registro entero, registro en coma flotante o registro de control.
  - Bit inicial (Int): define el bit de inicio del campo dentro de la instrucción.
  - Bit final (Int): indica el ultimo bit que ocupa este campo en la instrucción.
- Definición de sintaxis (String): define la sintaxis de la pseudoinstrucción indicando la posición de los diferentes campos, así como los diferentes símbolos que contenga entre estos.

- Sintaxis con tipos (String): muestra la sintaxis de la pseudoinstrucción indicando el tipo de dato que debe ir cada posición.
- Sintaxis con nombres (String): indica la sintaxis de la pseudoinstrucción mostrando el nombre del campo que se sitúa en cada posición.
- Definición de la pseudoinstrucción (String): indica las instrucciones que se tienen que ejecutar para realizar la función requerida.

Por último, las directivas, que son comandos que se utilizan para indicar al compilador un segmento, símbolos, subrutinas, etc. Los atributos que tienen las directivas son:

- Nombre (String): nombre de la directiva.
- Acción (String): indica la acción que realiza esta directiva. Estas acciones podrán ser:
  - Indicar inicio de un segmento, que podrá ser:
    - Segmento de código.
    - Segmento de datos.
  - Declarar una función como global.
  - Almacenar en la memoria un byte.
  - Guardar en la memoria media palabra.
  - Almacenar en la memoria una palabra.
  - Almacenar en la memoria una palabra doble.
  - Introducir en la memoria valores reales de precisión simple.
  - Guardar en la memoria valores reales de doble precisión.
  - Reservar espacio en la memoria.
  - Introducir en la memoria una cadena de caracteres.
  - Guardar en la memoria una cadena de caracteres, pero finalizando esta con un valor nulo.
  - Alinear la memoria.
- Tamaño (Int): se utiliza en algunas directivas para indicar el tamaño del dato que se introduce en la memoria.

4.2.2. Compilador

El compilador es un componente imprescindible para el funcionamiento del simulador, ya que se encarga de validar el código ensamblador introducido por el usuario y guardar en memoria las instrucciones a ejecutar, así como los datos. Este componente también permitirá generar un fichero con el código compilado en binario.

Si el código introducido es correcto, el compilador cargará en memoria todas las instrucciones y datos introducidos para que este código pueda ser simulado posteriormente, pero, en caso de encontrar un error, el compilador desplegará un aviso para informar al usuario del error que se ha producido y la línea de código donde se encuentra para que pueda subsanarlo. Por esto, se ha definido un conjunto de errores que se pueden producir y que el compilador es capaz de detectar, estos son:

- Etiqueta vacía.
- Etiqueta repetida.
- Directiva inválida.
- Directiva vacía.
- Dato incorrecto.
- Memoria no alineada.
- Instrucción no encontrada.
- Sintaxis de la instrucción incorrecta.
- Registro no encontrado.
- Valor inmediato demasiado grande.
- Valor inmediato no válido.
- Etiqueta de dirección de memoria inválida.
- Dirección de memoria demasiado grande.
- Dirección de memoria no válida.

- Definición de la pseudoinstruccion incorrecta.

La funcionalidad que permite generar un fichero con el binario será similar a la descrita anteriormente, salvo que en el fichero se guardarán las instrucciones escritas en binario y que permitirá que posteriormente se pueda cargar este fichero y ejecutar las instrucciones que tiene definidas llamándolas desde otro programa ensamblador.

4.2.3. Kernel de simulación

El kernel de simulación, como se mencionó anteriormente, se encarga de ejecutar el código ensamblador que el usuario ha introducido en la aplicación y que el compilador ha validado y almacenado en memoria.

Para realizar la ejecución, este componente debe tener en cuenta la definición de la arquitectura, ya que en la definición de cada una de las instrucciones se indican las operaciones que realiza la instrucción y es el atributo que el kernel de simulación utiliza para llevar a cabo su función. Además, también deberá hacer uso de la descripción de la arquitectura para comprobar que la instrucción puede leer y escribir en los registros que la instrucción utiliza en cada una de las acciones y, en el caso de que un registro no pueda ser leído o escrito, este componente se lo notificará al usuario y detendrá la ejecución del programa. Adicionalmente, este componente también será capaz de detectar si la definición de la instrucción es correcta o contiene errores sintácticos y, en este caso, también lo notificará al usuario y parará la ejecución.

Este componente podrá realizar la simulación del programa instrucción a instrucción o el programa completo, deteniendo la ejecución cuando esta llega al final o cuando encuentra un punto de ruptura.

4.2.4. Interfaz de usuario

La interfaz de usuario será el componente encargado de mostrar el estado actual del simulador al usuario, así como ser el componente sobre el que interacciona el usuario para realizar acciones sobre los componentes explicados anteriormente. La interfaz de usuario estará formada por tres pantallas diferentes que estarán relacionadas directamente con uno

de los componentes anteriores.

En la pantalla destinada a la simulación se podrán ver los diferentes registros que tiene la arquitectura y su valor actual en diferentes formatos. También se podrán ver los datos y las instrucciones que se encuentran almacenadas en la memoria principal del simulador. Las acciones que se pueden realizar sobre esta pantalla están relacionados con el kernel de simulación como es el caso de la ejecución instrucción a instrucción o la ejecución del programa completo.

La pantalla destinada al editor de la arquitectura está formada por diferentes vistas, cada una estará destinada a un tipo de característica de la arquitectura y se podrán ver los atributos que la define. En esta pantalla los usuarios podrán editar los atributos de las características ya definidas, borrar una característica o crear una nueva mediante distintos formularios. Además de estas acciones, también tendrá la posibilidad de guardar la arquitectura definida de forma local.

Por último, existe una tercera pantalla que se utilizará como entrada para el código ensamblador y, en ella, el usuario podrá escribir el código que desea compilar, cargarlo de un fichero que se encuentra almacenado de forma local, guardar el código introducido en un fichero y compilar el código introducido.

Capítulo 5

Implementación y despliegue

En este capítulo se va a detallar la implementación y el despliegue del simulador. Respecto a la implementación (Sección 5.1) de la herramienta, se van a desarrollar las secciones del código que han resultado más laboriosas y que, además, son más importantes para el funcionamiento del simulador. Además, se explicará cómo se debe llevar a cabo el despliegue (Sección 5.2) de la herramienta para que los usuarios puedan utilizarla.

5.1. Implementación

En esta sección se va a explicar, a alto nivel, como se ha realizado la implementación del simulador a partir de la solución final descrita en la sección 4.1.

Como ya se expuso en el Capítulo 4 Diseño, el simulador se va a desarrollar mediante los lenguajes de programación HTML, CSS y JavaScript, junto con los *frameworks* Bootstrap + Vue y jQuery.js.

Para lograr el funcionamiento deseado se han tenido que implementar tres módulos diferentes. Un módulo se encarga de la definición de la arquitectura, otro se ocupa de la compilación del código ensamblador y, por último, el módulo restante se encarga de ejecutar el código ensamblador definido por el usuario.

En consecuencia, a lo expuesto en las líneas anteriores, el simulador está formado por diferentes ficheros que se encargan de definir la interfaz de usuario, la funcionalidad y las arquitecturas disponibles. A continuación, se muestra un diagrama en el que se detalla

la estructura de ficheros que utiliza el simulador.

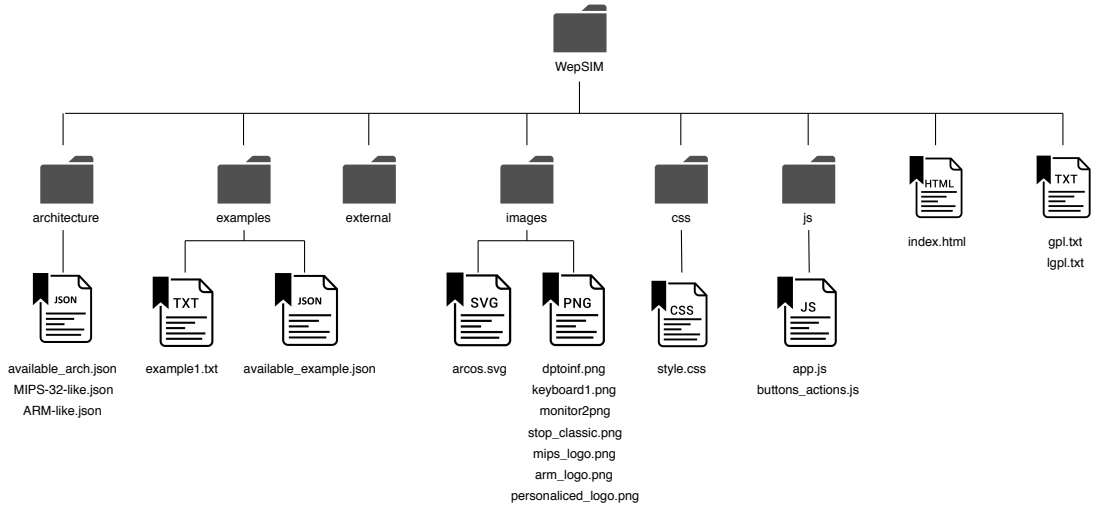


Fig. 5.1. Estructura de ficheros

Como se puede ver en la imagen 5.1 el simulador está formado por pocos ficheros, esto se debe a que Boostrap + Vue crea aplicaciones de una sola página, es decir, las aplicaciones que utilizan este *framework* están formadas por un fichero HTML y un fichero JavaScript que contienen toda la interfaz de usuario y funcionalidad respectivamente. A continuación, se van a explicar la función que lleva a cabo cada fichero en el funcionamiento de la herramienta.

- **architecture/available_arch.json:** este fichero indica las arquitecturas que están disponibles por defecto en el simulador, así como, el nombre de la arquitectura, una descripción de la arquitectura, la ruta al directorio en el que se encuentra el logotipo y la ruta al directorio donde se encuentra la definición de la arquitectura.

- **architecture/MIPS-32-like.json y architecture/ARM-like.json:** estos dos ficheros definen las características que tienen MIPS-32 y ARM, como pueden ser los registros, instrucciones, pseudoinstrucciones o directivas.

- **examples/available_example.json:** se encarga de indicar los ejemplos que están disponibles en el simulador y en él se indican el nombre del ejemplo, una descripción de las acciones que realiza y la ruta al directorio en el que se encuentra el ejemplo.

- **external:** este directorio contiene el código de los *frameworks* que utiliza el simulador, estos son: Bootstrap + Vue y jQuery.js.

- **images:** este directorio contiene las imágenes que se utilizan en la interfaz de usuario del simulador, como pueden ser el logotipo del departamento de informática, el logotipo del grupo de investigación ARCOS, el icono que se utiliza para mostrar los *breakpoints* de la ejecución, los iconos de la consola del simulador y los logotipos de las arquitecturas.

- **css/style.css:** este fichero contiene las diferentes definiciones de estilo que se utilizan en la interfaz de usuario del sistema.

- **js/app.js:** este fichero contiene las funcionalidades que se llevan a cabo mediante Bootstrap + Vue, como puede ser el funcionamiento de las acciones que se realizan en el editor de la arquitectura, la compilación de un programa ensamblador introducido por el usuario o la simulación de un programa.

- **js/buttons_actions.js:** este fichero se encarga de realizar las acciones que se producen con determinados botones de la interfaz de usuario como son los botones que se encargan de cambiar la pantalla que se muestra. Todas estas funciones se realizan mediante jQuery.js.

- **index.html:** este fichero tiene la función de crear la interfaz de usuario del simulador.

- **gpl.txt y lgpl.txt:** estos dos ficheros indican las condiciones de uso de las licencias que tiene este software.

A continuación, se va a detallar la implementación de las funciones que se encargan de realizar el compilado y la simulación del código introducido por el usuario y así entender el funcionamiento. La compilación del programa ensamblador que ha introducido el usuario se mostrará en los Algorithm 1 y 2 y, como esta funcionalidad es compleja, hace uso de otras dos funciones que se encargan de compilar el segmento de datos y el segmento de texto por separado, por ello en los Algorithm 3, 4 y 5 se explicará el proceso de compilación del segmento de datos y en los Algorithm 6, 7 y 8 el compilado del segmento de texto. En el Algorithm 9 se detallará el proceso que se tiene que seguir si se tiene que compilar una pseudoinstrucción y, por último, se explicará cómo se lleva a cabo la simulación del código compilado en los Algorithm 10 y 11.

Para compilar el programa en ensamblador introducido por el usuario lo primero que se realizará será reiniciar las estructuras de datos que utiliza el simulador para realizar la ejecución y crear una copia de seguridad del código ensamblador introducido y la arquitectura que se está utilizando. Después, se comprueba que hay disponible un programa para realizar el compilado y, posteriormente, se obtendrá el primer token (segmentos que se obtienen al seccionar el programa ensamblador introducido) que corresponderá con una directiva y se ejecutará el código correspondiente. Este proceso se realizará hasta que se encuentre una directiva que no existe o se termine el programa. Una vez compilado el programa se verifica que no existe solapamiento entre los diferentes segmentos de la memoria, que las instrucciones pendientes son correctas y, por último, se almacena en las estructuras de datos correspondientes los binarios de las funciones.

Algorithm 1: Función assemblyCompiler Parte 1

```
   /* Acciones previas a la compilación                              */
1  Se reinician las estructuras de datos del simulador
2  Se indican las direcciones de memoria de inicio de cada segmento
3  Se guarda en la caché del navegador una copia de seguridad del código y la
     arquitectura actual
   /* Compilación                                                    */
4  while Programa no terminado do
5      Se obtiene un token
6      if Token vacío then
7          Programa terminado
8          break
9      foreach Directiva do
10         if Token es igual a la directiva then
11             if La directiva es de segmento de datos then
12                 Se llama a la función dataSegmentCompiler()
```

Algorithm 2: Función assemblyCompiler Parte 2

```
13
14
15
16          if La directiva es de segmento de texto then
17              Se llama a la función textSegmentCompiler()
18          if La directiva es globl then
19              Se guardan las etiquetas que contiene la directiva
20          if La directiva es extern then
21              Se importa el código que incluye la etiqueta indicada
22      if Es la última directiva y el token no es válido then
23          Mostrar error
24          Se reinician las estructuras de datos del simulador
25          return error
26  if Se solapan los espacios de memoria then
27      Mostrar error
28      Se reinician las estructuras de datos del simulador
29      return error
30  foreach Instrucción pendiente do
31      if Etiqueta no existe en el segmento de datos then
32          Mostrar error
33          Se reinician las estructuras de datos del simulador
34          return error
35  foreach Binario do
36      Se almacena la función en una estructura de binarios
```

Para compilar el segmento de datos, en primer lugar, se tiene que obtener el token y comprobar que no esté vacío. Si este token es una etiqueta se almacena y se lee el siguiente token. Una vez realizados estos pasos se obtendrá una directiva y dependiendo del tipo de directiva que se trata, se realizarán unas acciones u otras para almacenar los datos en la

memoria o actualizar la alineación de la memoria, ya que el tipo de datos que manipulan cada una de las directivas son diferentes, aunque en la mayoría de las directivas se realizan acciones similares como la acción de alinear los datos en memoria.

Algorithm 3: Función dataSegmentCompiler Parte 1

```
1  while Existan datos en el segmento do
2      Se obtiene el token
3      if Token vacío then
4          break
5      if El token es una etiqueta then
6          if La etiqueta existe then
7              Mostrar error
8              return error
9          else
10             Se obtiene el siguiente token
11     foreach Directiva do
12         if El token es la directiva byte then
13             while Existan datos en la directiva do
14                 Se obtiene el token
15                 Se alinea la memoria según lo indicado
16                 Se guarda el dato en la memoria
17         if El token es la directiva half then
18             while Existan datos en la directiva do
19                 Se obtiene el token
20                 Se alinea la memoria según lo indicado
21                 Se guarda el dato en la memoria
```

Algorithm 4: Función dataSegmentCompiler Parte 2

```
22
23
24      if El token es la directiva word then
25          while Existan datos en la directiva do
26              Se obtiene el token
27              Se alinea la memoria según lo indicado
28              Se guarda el dato en la memoria
29      if El token es la directiva doubleword then
30          while Existan datos en la directiva do
31              Se obtiene el token
32              Se alinea la memoria según lo indicado
33              Se guarda el dato en la memoria
34      if El token es la directiva float then
35          while Existan datos en la directiva do
36              Se obtiene el token
37              Se alinea la memoria según lo indicado
38              Se guarda el dato en la memoria
39      if El token es la directiva double then
40          while Existan datos en la directiva do
41              Se obtiene el token
42              Se alinea la memoria según lo indicado
43              Se guarda el dato en la memoria
44      if El token es la directiva ascii then
45          while Existan datos en la directiva do
46              while No se termine la cadena de caracteres do
47                  Se obtiene el token
48                  Se concatenan los tokens de la cadena
49              Se alinea la memoria según lo indicado
50              Se guarda el dato en la memoria
```

Algorithm 5: Función dataSegmentCompiler Parte 3

```
51
52
53      if El token es la directiva asciiz then
54          while Existan datos en la directiva do
55              while No se termine la cadena de caracteres do
56                  Se obtiene el token
57                  Se concatenan los tokens de la cadena
58              Se añade un espacio en blanco al final de la cadena de caracteres
59              Se alinea la memoria según lo indicado
60              Se guarda el dato en la memoria
61      if El token es la directiva space then
62          Se lee el número de bytes a reservar
63          Se alinea la memoria según lo indicado
64          Se reserva el espacio en memoria
65      if El token es la directiva align then
66          Se lee el nuevo valor que se va a usar para alinear la memoria
67          Se modifica la variable que indica la alineación
```

Para compilar el segmento de texto, primero se obtiene un nuevo token y se verifica que no se trata de una directiva, si es así, se compilan las instrucciones pendientes y se genera una tabla con las etiquetas de las instrucciones. Si no es una directiva, se comprueba si el token es una etiqueta, ya que en caso de serlo se almacenará y se obtendrá el siguiente token. El siguiente paso es recorrer las instrucciones que están definidas en la arquitectura y comparar el token actual con el nombre de cada una de las instrucciones de la arquitectura. Si el nombre es igual, quiere decir que la instrucción está definida en la arquitectura y, por lo tanto, se cuenta el número de campos que tiene esta instrucción para posteriormente realizar un bucle en el que se verificarán los campos introducidos por el usuario; si esta verificación es correcta la instrucción es almacenada en memoria y, en el caso contrario, se deberá verificar si esta instrucción es una pseudoinstrucción. En caso de serlo, se llama a la función encargada de compilar una pseudoinstrucción y en caso

contrario se muestra un error.

Algorithm 6: Función textSegmentCompiler Parte 1

```
1  while Existan instrucciones en el segmento do
2      Se obtiene el token
3      foreach Directiva do
4          if El token es igual a la directiva then
5              Se compilan las instrucciones pendientes
6              Se genera la tabla de etiquetas de las instrucciones
7              return
8      if El token es una etiqueta then
9          if La etiqueta existe then
10             Mostrar error
11             return error
12         else
13             Se obtiene el siguiente token
14     foreach Intrucción do
15         if Token es distinto del nombre de la instrucción then
16             continue
17         else
18             Se cuenta el número de campos que tiene la instrucción
19             foreach Campo de la instrucción do
20                 Se obtiene un nuevo token
21                 if El campo es un registro then
22                     foreach Registro do
23                         if El registro de la instrucción es igual al nombre o
                             indice del registro then
24                             Se indica que es válido
25                         if Último registro y no es válido then
26                             Mostrar error
27                             return error
```

Algorithm 7: Función textSegmentCompiler Parte 2

```
28
29
30
31
32  if El campo es un valor inmediato then
33      if El tamaño del valor inmediato es mayor que el del campo then
34          if Existe pseudointrucción then
35              Se compila la pseudointrucción
36          else
37              Mostrar error
38              return error
39      if El valor inmediato es una etiqueta then
40          foreach Etiqueta do
41              if Es la última etiqueta y no es igual al valor inmediato then
42                  Se encola en la lista de instrucciones pendientes de compilar
43              if Es la última etiqueta, no es igual al valor inmediato y es una instrucción pendiente then
44                  Mostrar error
45                  return error
46              if La etiqueta es igual al valor inmediato then
47                  Se indica que es válido
```

Algorithm 8: Función textSegmentCompiler Parte 3

```
48
49
50
51
52      if El campo es una dirección de memoria then
53          if El tamaño de la dirección de memoria es mayor que el del
             campo then
54              Mostrar error
55              return error
56          if La direccion de memoria es una etiqueta then
57              foreach Etiqueta do
58                  if Es la última etiqueta y no es igual a la dirección
                     de memoria then
59                      Se encola en la lista de instrucciones pendientes
                         de compilar
60                  if Es la última etiqueta, no es igual a la dirección de
                     memoria y es una instrucción pendiente then
61                      Mostrar error
62                      return error
63                  if La etiqueta es igual al dirección de memoria then
64                      Se indica que es válido
65      Se almacena la instrucción en memoria
66  if Token no coincide con el nombre de ninguna instrucción then
67      foreach Pseudoinstrucción do
68          if Token igual al nombre de la pseudoinstrucción then
69              Se compila la pseudointrucción
70          if Es la última pseudoinstrucción y no coincide el nombre then
71              Mostrar error
72              return error
```

Para compilar una pseudointrucción, en primer lugar, se obtienen los campos que la forman y, posteriormente, se recorren todas las pseudoinstrucciones y si el nombre de la pseudoinstrucción es igual que el del primer campo se obtiene la definición y de ella las instrucciones que se ejecutan para realizar la funcionalidad de esta pseudoinstrucción. Después, cada una de estas instrucciones se deberán encolar en la lista de instrucciones pendientes para que sean compiladas mediante el algoritmo explicado anteriormente.

Algorithm 9: Función pseudoinstructionCompiler Parte 1

```
1  Se obtienen los campos de la pseudoinstrucción
2  foreach Pseudoinstrucción do
3      if Nombre de la pseudoinstrucción es distinto al primer campo then
4          continue
5      else
6          if Sintaxis incorrecta then
7              Mostrar error
8              return error
9          Se obtiene la definicion de la pseudoinstrucción
10         Se obtienen las instrucciones que forman la definición
11         foreach Intrucción de la definición do
12             Se encola la instrucción en la lista de instrucciones pendientes de
                compilar
13     if El primer campo no coincide con el nombre de ninguna instrucción then
14         Mostrar error
15         return error
```

Para realizar la simulación del programa compilado, en primer lugar, hay que comprobar que existen instrucciones guardadas en la memoria o si el programa ya ha finalizado, una vez realizadas esas comprobaciones se debe buscar la etiqueta “main” que indica el inicio del programa. Cuando ya se ha localizado la primera instrucción que se tiene que ejecutar se obtiene la sintaxis de la instrucción, ya que esta nos permite saber cómo están estructurados los campos que la forman, así como el tipo y nombre de cada uno. Después se incrementa en cuatro el registro “PC” y se obtiene la definición de la instrucción ya que nos indica las acciones que realiza la instrucción y que campos utiliza para

realizarlas. Por ello, posteriormente, se sustituyen los registros por funciones que leen o escriben en el registro indicado en la definición y, para las llamadas al sistema y accesos a memoria, se sigue un proceso similar al de los registros que permite ejecutar ese código y realizar las acciones de la definición. Este proceso se sigue hasta que se produzca un error o se termine el programa, actualizando la tabla de estadísticas en cada ejecución.

Algorithm 10: Función executeInstruction Parte 1

```
   /* Verificaciones anteriores a la ejecución                    */
1  if No hay instrucciones cargadas en memoria then
2      Mostrar error
3      return error
4  if Programa finalizado then
5      Mostrar error
6      return error
7  if Programa finalizado con errores then
8      Mostrar error
9      return error
   /* Se localiza la etiqueta main que indica el inicio del programa */
10 foreach Instrucción do
11     if La instrucción tiene la etiqueta main then
12         PC igual a la dirección de memoria de la instrucción
13         Se sale del bucle
14     if Es la última instrucción y no se ha encontrado then
15         Mostrar error
16         return error
```

Algorithm 11: Función executeInstruction Parte 2

```
/* Se ejecuta la instrucción                                    */
foreach Instrucción definida do
    if Nombre instrucción a ejecutar es igual a la instrucción definida actual
     then
        Se obtiene la definición de la sintaxis
        Se construye una expresión regular para obtener los distintos campos
        Se genera la sintaxis con los tipos de los campos
        Se genera la sintaxis con los nombres de los campos
        Se guarda en una variable la definición de la instrucción
        Se incrementa el PC en 4
        Se obtienen los campos de la instrucción introducida por el usuario
        Se remplazan los valores de la definición por los nombres de los
         registros
        if Es una llamada al sistema then
            Se remplaza la llamada al sistema por la funcion syscall()
        foreach Registro do
            if Existe el registro en la definición y se escribe en él then
                Se remplaza el nombre del registro por writeRegister()
            if Existe el registro en la definición y se lee de él then
                Se remplaza el nombre del registro por readRegister()
        if Se escribe en memoria then
            Se remplaza la instrucción que lo indica por writeMemory()
        if Se lee de memoria then
            Se remplaza la instrucción que lo indica por readMemory
        Se ejecuta la definición de la instrucción con todos estos cambios
         realizados.
        if Error al ejecutar then
            Mostrar error
            return error
        Se actualizan las estadísticas de ejecución
        Se comprueba si ha terminado el programa.
```

5.2. Despliegue

En este apartado se va a detallar cómo se debe desplegar el simulador y, para ello, se van a indicar unos requisitos recomendados, cómo se debe realizar el despliegue de la herramienta para que pueda ser utilizada y cómo se debe ejecutar.

Los requisitos que deben cumplir los dispositivos para que el usuario tenga una experiencia satisfactoria cuando utilice la herramienta deben ser:

- **Sistema Operativo:** Windows 10, Ubuntu 18.04 LTS, MacOS 10.14, Android 6, iOS 11.
- **Procesador:** Intel(R) Core(TM) i3 CPU 8100 @3.6GHz o superior.
- **Memoria RAM:** 2 GB o superior.
- **Almacenamiento:** 1 GB de espacio libre en el disco duro (espacio recomendado para los navegadores web).
- **Red:** no es necesario estar conectado a internet para utilizar el simulador, pero si para acceder a él.
- **software:** los navegadores web recomendados para utilizar el simulador son:
  - Google Chrome 50+
  - Mozilla Firefox 45+
  - Microsoft Edge 30+
  - Safari 10+

Para desplegar el simulador será necesario tener instalado en el ordenador del usuario un servidor web *Apache* para poder acceder a la herramienta. Después de instalar el servidor web en el ordenador se tendrán que realizar los siguientes pasos para almacenar la herramienta en el servidor:

1. Se crea un directorio en el servidor web que se ocupará de albergar el simulador.
2. Se accede al directorio creado en el paso anterior.
3. Se clona el repositorio en el que se encuentra el simulador para obtener el código fuente de la herramienta, para ello se utiliza el siguiente comando:

   git clone https://github.com/dcamarmas/creator.git
4. Se reinicia el servicio del servidor web para que se realicen correctamente los cambios y esté disponible el simulador.

Para utilizar la herramienta el usuario deberá utilizar un navegador web al que le indicará la dirección en la que está alojado el simulador.

A su vez el simulador también está disponible en la dirección:

`https://dcamarmas.github.io/creator/`

Esto permite que el usuario no tenga que realizar el proceso anterior.

Capítulo 6

Verificación, validación y evaluación

En este capítulo se va a describir la verificación, validación y evaluación del simulador. Se detallará la verificación y la validación (Sección 6.1), en las que se expondrán un conjunto de pruebas que permiten comprobar que se satisfacen los requisitos especificados en el Capítulo 3 Análisis, así como comprobar que las simulaciones realizadas por la herramienta son correctas y se obtienen los resultados esperados. Además, también se va a realizar una evaluación (Sección 6.2) de la herramienta desarrollada frente a otro simulador existente.

6.1. Verificación y validación

En esta sección se va a comprobar que los requisitos que se definieron en el Capítulo 3 están reflejados en el sistema. Además de verificar que se cumplen los requisitos definidos, con estas pruebas también se pretende comprobar que los resultados que se obtienen con una simulación de un programa ensamblador son los esperados y para ello se utiliza el libro de la asignatura Estructura de Computadores [32].

Se van a realizar dos tipos de comprobaciones en esta sección, la primera será la Verificación Software (Sección 6.1.1). Esta verificación consiste en determinar si se cumplen los requisitos software durante el desarrollo de la herramienta. La segunda comprobación que se realizará será la Validación Software (Sección 6.1.2) que se realiza, una vez implementado el sistema, para comprobar que se cumplen los requisitos de usuario que determinan el funcionamiento final de la herramienta.

6.1.1. Pruebas de verificación

Durante el desarrollo del sistema será necesario realizar diferentes pruebas para comprobar que la herramienta se está desarrollando según lo establecido por los requisitos software, por lo tanto, para definir las diferentes pruebas de verificación se va a utilizar la siguiente plantilla (Tabla 6.1). El campo "identificador" tendrá el formato PVE-XX, siendo XX el número de la prueba de verificación.

TABLA 6.1. PLANTILLA PRUEBAS DE VERIFICACIÓN

Identificador	Identificador de la prueba.
Nombre	Nombre de la prueba.
Requisitos	Requisitos software que verifica esta prueba.
Descripción	Explicación de la prueba realizada.
Precondiciones	Requisitos que se tienen que cumplir para poder realizar la prueba.
Procedimiento	Pasos que se tienen que ejecutar para realizar la prueba.
Postcondiciones	Condiciones que se tienen que cumplir obligatoriamente después de realizar la prueba.
Evaluación	OK si la verificación es correcta o Error en caso contrario.

A continuación, se muestran las pruebas de verificación realizadas:

TABLA 6.2. PRUEBA DE VERIFICACIÓN PVE-01

Identificador	PVE-01
Nombre	Simulador genérico.
Requisitos	RW-F-F01, RW-F-F03 y RW-F-F04
Descripción	Se verifica que el simulador es capaz de simular el comportamiento de diferentes arquitecturas.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Cargar una arquitectura. 2. Cargar un programa ensamblador. 3. Compilar el programa. 4. Ejecutar la simulación. 5. Verificar que los resultados son correctos.
Postcondiciones	El programa es compilado correctamente y los resultados obtenidos en la simulación son correctos.
Evaluación	OK

TABLA 6.3. PRUEBA DE VERIFICACIÓN PVE-02

Identificador	PVE-02
Nombre	Elección de la arquitectura.
Requisitos	RW-F-F02
Descripción	Se verifica que se puede elegir una arquitectura determinada en el menú inicial.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Seleccionar la arquitectura deseada.
Postcondiciones	La arquitectura seleccionada se carga correctamente.
Evaluación	OK

TABLA 6.4. PRUEBA DE VERIFICACIÓN PVE-03

Identificador	PVE-03
Nombre	Añadir nuevas arquitecturas.
Requisitos	RW-F-F05
Descripción	Se verifica que se puede añadir una nueva definición de arquitectura al simulador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Disponer de la definición de una arquitectura en un fichero.
Procedimiento	1. Pulsar el botón “Load Architecture”. 2. Introducir un nombre para la arquitectura. 3. Seleccionar el fichero que contiene la definición. 4. Pulsar el botón “Ok”. 5. Seleccionar la arquitectura cargada en el menú.
Postcondiciones	La arquitectura se carga correctamente en el simulador.
Evaluación	OK

TABLA 6.5. PRUEBA DE VERIFICACIÓN PVE-04

Identificador	PVE-04
Nombre	Borrar definición de arquitectura.
Requisitos	RW-F-F06
Descripción	Se verifica que se puede eliminar la definición de una arquitectura del simulador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargada una nueva arquitectura en el simulador.
Procedimiento	1. Pulsar el botón “Delete” de la arquitectura que se desea borrar. 2. Confirmar el borrado.
Postcondiciones	La arquitectura borrada ya no aparece en el menú de arquitecturas.
Evaluación	OK

TABLA 6.6. PRUEBA DE VERIFICACIÓN PVE-05

Identificador	PVE-05
Nombre	Guardar arquitectura.
Requisitos	RW-F-F07
Descripción	Se verifica que se puede guardar la definición de la arquitectura cargada en el simulador de forma local.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Pulsar el botón “Architecture”. 2. Pulsar el botón “Save”. 3. Introducir el nombre del fichero. 4. Aceptar.
Postcondiciones	Se guarda de forma local un fichero con la definición de la arquitectura.
Evaluación	OK

TABLA 6.7. PRUEBA DE VERIFICACIÓN PVE-06

Identificador	PVE-06
Nombre	Restablecer Arquitectura.
Requisitos	RW-F-F08
Descripción	Se verifica que se pueden restablecer la definición de la arquitectura a los valores iniciales.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Pulsar el botón “Architecture”. 2. Pulsar el botón “Reset” de la característica que se desea restablecer. 3. Aceptar.
Postcondiciones	Se restablece la característica a los valores iniciales.
Evaluación	OK

TABLA 6.8. PRUEBA DE VERIFICACIÓN PVE-07

Identificador	PVE-07
Nombre	Editar Arquitectura.
Requisitos	RW-F-F09, RW-F-F10, RW-F-F11, RW-F-F12, RW-F-F13, RW-F-F14, RW-F-F15, RW-F-F16 y RW-F-F17,
Descripción	Se verifica que la arquitectura cargada en el simulador puede ser modificada.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Pulsar el botón “Architecture”. 2. Crear, modificar y borrar un registro. 3. Crear, modificar y borrar una instrucción. 4. Crear, modificar y borrar una pseudoinstrucción.
Postcondiciones	Los cambios se realizan correctamente, creándose una nueva definición de arquitectura.
Evaluación	OK

TABLA 6.9. PRUEBA DE VERIFICACIÓN PVE-08

Identificador	PVE-08
Nombre	Código ensamblador.
Requisitos	RW-F-F018, RW-F-F19 y RW-F-F20
Descripción	Se verifica que el usuario puede introducir y modificar el código ensamblador que se va a ejecutar.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Escribir o editar el código ensamblador.
Postcondiciones	El código introducido se muestra en la herramienta.
Evaluación	OK

TABLA 6.10. PRUEBA DE VERIFICACIÓN PVE-09

Identificador	PVE-09
Nombre	Cargar y guardar código ensamblador.
Requisitos	RW-F-F021 y RW-F-F22
Descripción	Se verifica que se puede cargar un código ensamblador y, también, guardarle de forma local.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura. 3. Tener en un fichero .txt un programa ensamblador.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Pulsar el botón “Load”. 3. Seleccionar el fichero que contiene el programa ensamblador. 4. Pulsar el botón “Save”. 5. Se introduce el nombre del fichero en el que se va a guardar el código. 6. Se acepta.
Postcondiciones	El código que se carga en el simulador es el mismo que contiene el fichero seleccionado y el código que contiene el fichero en el que se guarda el código ensamblador de forma local es igual que el que se ha introducido en el simulador.
Evaluación	OK

TABLA 6.11. PRUEBA DE VERIFICACIÓN PVE-10

Identificador	PVE-10
Nombre	Compilar programa.
Requisitos	RW-F-F023 y RW-F-F024
Descripción	Se verifica que el programa introducido se compila correctamente.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Escribir un programa ensamblador. 3. Compilar el programa.
Postcondiciones	El código disponible para ejecutar es el que se compiló anteriormente.
Evaluación	OK

TABLA 6.12. PRUEBA DE VERIFICACIÓN PVE-11

Identificador	PVE-11
Nombre	Importar y exportar una biblioteca.
Requisitos	RW-F-F025 y RW-F-F026
Descripción	Se verifica que se puede exportar e importar una biblioteca en el simulador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura. 3. Tener un programa ensamblador escrito.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Pulsar el botón “Save Library”. 3. Introducir el nombre del fichero en el que se va a guardar y aceptar. 4. Pulsar el botón “Load Library”. 5. Seleccionar el fichero que contiene la biblioteca que se quiere cargar y aceptar.
Postcondiciones	Se guarda y se carga la biblioteca correctamente.
Evaluación	OK

TABLA 6.13. PRUEBA DE VERIFICACIÓN PVE-12

Identificador	PVE-12
Nombre	Simulación.
Requisitos	RW-F-F027, RW-F-F028, RW-F-F029, RW-F-F030, RW-F-F031, RW-F-F032 y RW-F-F033
Descripción	Se verifica que la simulación de un programa se realiza correctamente.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura. 3. Tener un programa ensamblador compilado.
Procedimiento	1. Introducir un *breakpoint* en una instrucción. 2. Pulsar el botón “Run” para ejecutar el programa hasta el *breakpoint*. 3. Pulsar el botón “Inst.” para ejecutar el resto de las instrucciones paso a paso.
Postcondiciones	El programa se ejecuta correctamente mostrando el estado del simulador después de ejecutar cada instrucción.
Evaluación	OK

TABLA 6.14. PRUEBA DE VERIFICACIÓN PVE-13

Identificador	PVE-13
Nombre	Modificación del valor de un registro.
Requisitos	RW-F-F034 y RW-F-F035
Descripción	Se verifica que los valores actuales de los registros pueden ser modificados.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Seleccionar el registro que se desea modificar. 2. Introducir el nuevo valor en decimal. 3. Introducir el nuevo valor en hexadecimal.
Postcondiciones	El valor del registro se modifica correctamente en ambos casos.
Evaluación	OK

TABLA 6.15. PRUEBA DE VERIFICACIÓN PVE-14

Identificador	PVE-14
Nombre	Reinicio de la simulación.
Requisitos	RW-F-F036
Descripción	Se verifica que la simulación se puede reiniciar.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura. 3. Tener un programa ensamblador compilado.
Procedimiento	1. Ejecutar el programa completo. 2. Pulsar el botón “Reset”.
Postcondiciones	Se reinicia la ejecución y los registros y la memoria tienen los valores iniciales.
Evaluación	OK

TABLA 6.16. PRUEBA DE VERIFICACIÓN PVE-15

Identificador	PVE-15
Nombre	Registro de notificaciones.
Requisitos	RW-F-F037
Descripción	Se verifica que se registran correctamente las notificaciones que se muestran en el simulador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Cargar una arquitectura. 2. Pulsar el botón “Info”. 3. Pulsar el botón “Show Notifications”.
Postcondiciones	Las notificaciones se han registrado correctamente.
Evaluación	OK

TABLA 6.17. PRUEBA DE VERIFICACIÓN PVE-16

Identificador	PVE-16
Nombre	Copia de seguridad.
Requisitos	RW-F-F038, RW-F-F039 y RW-F-F040
Descripción	Se verifica que se guarda y carga correctamente una copia de seguridad de la arquitectura y del programa ensamblador introducido.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Escribir un programa ensamblador. 2. Compilar el programa ensamblador introducido. 3. Recargar la página web que contiene el simulador. 4. Cargar la copia de seguridad.
Postcondiciones	El simulador se encuentra en el mismo estado en que se encontraba cuando se compiló el programa antes de recargar la página.
Evaluación	OK

TABLA 6.18. PRUEBA DE VERIFICACIÓN PVE-17

Identificador	PVE-17
Nombre	Tamaño máximo de la arquitectura.
Requisitos	RW-F-I01
Descripción	Se verifica que no se puede crear una arquitectura de más de 64 bits.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Modificar el tamaño de la arquitectura a un valor mayor de 64 bits.
Postcondiciones	El simulador muestra un error.
Evaluación	OK

TABLA 6.19. PRUEBA DE VERIFICACIÓN PVE-18

Identificador	PVE-18
Nombre	Memoria alineada.
Requisitos	RW-F-I02
Descripción	Se verifica que no se pueden almacenar datos en memoria si estos no están alineados.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Escribir un programa ensamblador que almacene datos en memoria y que no estén alineados. 2. Compilar el programa.
Postcondiciones	El simulador muestra un error.
Evaluación	OK

TABLA 6.20. PRUEBA DE VERIFICACIÓN PVE-19

Identificador	PVE-19
Nombre	Plataforma.
Requisitos	RW-NF-PL01, RW-NF-PL02, RW-NF-PL03, RW-NF-PL04, RW-NF-PL5 y, RW-NF-PL06
Descripción	Se verifica que el simulador puede ser utilizado en diferentes plataformas.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La simulación se realiza correctamente.
Evaluación	OK

TABLA 6.21. PRUEBA DE VERIFICACIÓN PVE-20

Identificador	PVE-20
Nombre	Conexión a internet.
Requisitos	RW-NF-PL07 y RW-NF-PL08
Descripción	Se verifica que la herramienta funciona correctamente sin conexión a internet.
Precondiciones	1. No tener conexión a internet. 2. Tener en el dispositivo el código fuente del simulador. 3. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 4. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La simulación se realiza correctamente.
Evaluación	OK

TABLA 6.22. PRUEBA DE VERIFICACIÓN PVE-21

Identificador	PVE-21
Nombre	Tecnología de desarrollo.
Requisitos	RW-NF-PL09
Descripción	Se verifica que la herramienta se ha desarrollado mediante HTML5.
Precondiciones	1. Tener en el dispositivo el código fuente del simulador.
Procedimiento	1. Verificar el código de los ficheros que forman el simulador
Postcondiciones	La herramienta se ha desarrollado mediante HTML5.
Evaluación	OK

TABLA 6.23. PRUEBA DE VERIFICACIÓN PVE-22

Identificador	PVE-22
Nombre	Interfaz de usuario.
Requisitos	RW-NF-UI01 y RW-NF-UI02
Descripción	Se verifica que la interfaz de usuario es compatible con PCs y con dispositivos móviles.
Precondiciones	1. Tener un dispositivo móvil y un PC. 2. Ejecutar el simulador en un navegador web igual o superior a los indicados. 3. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La interfaz de usuario se visualiza correctamente tanto en los dispositivos móviles y en los ordenadores.
Evaluación	OK

TABLA 6.24. PRUEBA DE VERIFICACIÓN PVE-23

Identificador	PVE-23
Nombre	Tiempo de ejecución de una instrucción.
Requisitos	RW-NF-R01
Descripción	Se verifica que el tiempo medio que tarda en ejecutarse una instrucción no es mayor de 0.5 segundos.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa midiendo el tiempo que tarda en ejecutarse cada instrucción.
Postcondiciones	El tiempo medio de ejecución de las instrucciones no sobrepasa los 0.5 segundos.
Evaluación	OK

A continuación, se muestra una tabla en la que se puede observar que todos los requisitos software han sido verificados.

TABLA 6.25. MATRIZ DE TRAZABILIDAD PRUEBAS DE VERIFICACIÓN

Requisitos	PVE-01	PVE-02	PVE-03	PVE-04	PVE-05	PVE-06	PVE-07	PVE-08	PVE-09	PVE-10	PVE-11	PVE-12	PVE-13	PVE-14	PVE-15	PVE-16	PVE-17	PVE-18	PVE-19	PVE-20	PVE-21	PVE-22	PVE-23
RW-F-F01	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F02	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F03	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F04	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F05	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F06	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F07	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F08	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F09	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F10	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F11	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F12	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F13	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F14	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F15	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F16	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F17	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F18	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F19	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F20	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F21	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F22	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F23	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F24	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F25	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F26	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
RW-F-F27	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F28	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F29	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F30	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F31	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F32	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F33	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RW-F-F34	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RW-F-F35	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RW-F-F36	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
RW-F-F37	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RW-F-F38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
RW-F-F39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
RW-F-F40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
RW-F-I01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RW-F-I02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
RW-NF-PL01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL03	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL05	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL06	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RW-NF-PL07	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
RW-NF-PL08	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
RW-NF-PL09	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
RW-NF-UI01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-
RW-NF-UI02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-
RW-NF-R01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓

6.1.2. Pruebas de validación

Una vez terminado el desarrollo del sistema se debe comprobar si su funcionamiento es el indicado por el usuario al inicio del proyecto, para ello, se van a definir diferentes pruebas de validación. Estas pruebas utilizarán la siguiente plantilla (Tabla 6.26). El campo ”identificador” tendrá el formato PVA-XX, siendo XX el número de la prueba de validación.

TABLA 6.26. PLANTILLA PRUEBAS DE VALIDACIÓN

Identificador	Identificador de la prueba.
Nombre	Nombre de la prueba.
Requisitos	Requisitos de usuario que verifica esta prueba.
Descripción	Explicación de la prueba realizada.
Precondiciones	Requisitos que se tienen que cumplir para poder realizar la prueba.
Procedimiento	Pasos que se tienen que ejecutar para realizar la prueba.
Postcondiciones	Condiciones que se tienen que cumplir obligatoriamente después de realizar la prueba.
Evaluación	OK si la validación es correcta o Error en caso contrario.

Las pruebas de validación realizadas son:

TABLA 6.27. PRUEBA DE VALIDACIÓN PVA-01

Identificador	PVA-01
Nombre	Simulador genérico.
Requisitos	RU-C01 y RU-C03
Descripción	Se valida que el simulador es capaz de simular el comportamiento de diferentes arquitecturas.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Cargar una arquitectura. 2. Escribir un programa ensamblador. 3. Compilar el programa. 4. Ejecutar la simulación. 5. Verificar que los resultados son correctos.
Postcondiciones	El programa es compilado correctamente y los resultados obtenidos en la simulación son correctos.
Evaluación	OK

TABLA 6.28. PRUEBA DE VALIDACIÓN PVA-02

Identificador	PVA-02
Nombre	Elección de la arquitectura.
Requisitos	RU-C02
Descripción	Se valida que se puede elegir una arquitectura determinada en el menú inicial.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Seleccionar la arquitectura deseada.
Postcondiciones	La arquitectura seleccionada se carga correctamente.
Evaluación	OK

TABLA 6.29. PRUEBA DE VALIDACIÓN PVA-03

Identificador	PVA-03
Nombre	Operaciones con las arquitecturas disponibles.
Requisitos	RU-C04
Descripción	Se valida que se puede añadir y borrar arquitecturas del menú.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener definida una arquitectura en un fichero.
Procedimiento	1. Pulsar el botón “Load Architecture”. 2. Introducir el nombre que se le quiere dar a la nueva arquitectura. 3. Seleccionar el fichero que contiene la definición de la nueva arquitectura y aceptar. 4. Pulsar el botón “Delete” de la arquitectura que se desea borrar.
Postcondiciones	La nueva arquitectura se añade y se borra correctamente.
Evaluación	OK

TABLA 6.30. PRUEBA DE VALIDACIÓN PVA-04

Identificador	PVA-04
Nombre	Operaciones con la arquitectura cargada.
Requisitos	RU-C05
Descripción	Se valida que se puede guardar la definición de la arquitectura actual, así como que esta se puede reiniciar a los valores iniciales.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Pulsar el botón “Architecture”. 2. Pulsar el botón “Save”. 3. Introducir el nombre del fichero que va a contener la definición de la arquitectura y aceptar. 4. Pulsar el botón “Reset” de las características que tiene la arquitectura cargada en el simulador y aceptar.
Postcondiciones	El fichero guardado de forma local contiene la definición de la arquitectura y el reinicio de la arquitectura se realiza correctamente.
Evaluación	OK

TABLA 6.31. PRUEBA DE VALIDACIÓN PVA-05

Identificador	PVA-05
Nombre	Operaciones con la arquitectura cargada.
Requisitos	RU-C06 y RU-C07
Descripción	Se valida que se pueden editar las diferentes características que forman la definición de la arquitectura.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Pulsar el botón “Architecture”. 2. Crear, modificar y borrar un registro. 3. Crear, modificar y borrar una instrucción. 4. Crear, modificar y borrar una pseudoinstrucción.
Postcondiciones	Los cambios se realizan correctamente y se crea una nueva definición de arquitectura.
Evaluación	OK

TABLA 6.32. PRUEBA DE VALIDACIÓN PVA-06

Identificador	PVA-06
Nombre	Definición del código ensamblador.
Requisitos	RU-C08
Descripción	Se valida que el usuario puede definir un código ensamblador que sigue el formato de las instrucciones definidas en la arquitectura cargada en el simulador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Escribir un programa en ensamblador.
Postcondiciones	El programa escrito se muestra correctamente en el simulador.
Evaluación	OK

TABLA 6.33. PRUEBA DE VALIDACIÓN PVA-07

Identificador	PVA-07
Nombre	Operaciones con el código ensamblador.
Requisitos	RU-C09
Descripción	Se valida que se puede cargar y guardar un código ensamblador.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador. 3. Tener definido un código ensamblador en un fichero.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Pulsar el botón “Load”. 3. Seleccionar el fichero que contiene el código ensamblador y aceptar. 4. Pulsar el botón “Save”. 5. Introducir el nombre con el que se guarda el fichero que contiene el código y aceptar.
Postcondiciones	El programa ensamblador se carga y guarda correctamente.
Evaluación	OK

TABLA 6.34. PRUEBA DE VALIDACIÓN PVA-08

Identificador	PVA-08
Nombre	Compilado del código ensamblador.
Requisitos	RU-C10
Descripción	Se valida que el simulador compila correctamente el código ensamblador introducido.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Escribir un programa en ensamblador. 3. Pulsar el botón “Compile”.
Postcondiciones	El programa se compila correctamente y el simulador esta preparado para realizar la simulación si no ha encontrado fallos en el código introducido.
Evaluación	OK

TABLA 6.35. PRUEBA DE VALIDACIÓN PVA-09

Identificador	PVA-09
Nombre	Operaciones con el binario del código ensamblador.
Requisitos	RU-C11
Descripción	Se valida que el simulador permite cargar y guardar una biblioteca.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Pulsar el botón “Assembly”. 2. Escribir un programa en ensamblador. 3. Pulsar el botón “Save Library”. 4. Pulsar el botón “Load Library”.
Postcondiciones	Se genera y se guarda la biblioteca en un fichero y, además, se carga correctamente.
Evaluación	OK

TABLA 6.36. PRUEBA DE VALIDACIÓN PVA-10

Identificador	PVA-10
Nombre	Simulación.
Requisitos	RU-C12, RU-C13 y RU-C14
Descripción	Se valida que la herramienta realiza las simulaciones correctamente.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador. 3. Tener un programa ensamblador compilado.
Procedimiento	1. Introducir un *breakpoint* en una instrucción. 2. Pulsar el botón "Run" para ejecutar el programa hasta el *breakpoint*. 3. Ejecutar las instrucciones restantes paso a paso pulsando el botón "Inst".
Postcondiciones	El programa se ejecuta correctamente y además se muestra el estado del simulador después de ejecutar cada instrucción.
Evaluación	OK

TABLA 6.37. PRUEBA DE VALIDACIÓN PVA-11

Identificador	PVA-11
Nombre	Modificación del valor de un registro.
Requisitos	RU-C15
Descripción	Se valida que se puede modificar el valor almacenado en un registro.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Seleccionar el registro que se quiere modificar. 2. Introducir un valor en decimal y aceptar. 3. Introducir un valor en hexadecimal y aceptar.
Postcondiciones	El valor se modifica correctamente en ambos casos.
Evaluación	OK

TABLA 6.38. PRUEBA DE VALIDACIÓN PVA-12

Identificador	PVA-12
Nombre	Reiniciar la simulación.
Requisitos	RU-C16
Descripción	Se valida que se puede modificar el valor almacenado en un registro.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador. 3. Tener un programa ensamblador compilado.
Procedimiento	1. Ejecutar la simulación del programa completo. 2. Pulsar el botón “Reset”.
Postcondiciones	Los valores de los registros y de la memoria vuelven a su estado inicial, así como la simulación que vuelve al inicio del programa.
Evaluación	OK

TABLA 6.39. PRUEBA DE VALIDACIÓN PVA-13

Identificador	PVA-13
Nombre	Registro de notificaciones.
Requisitos	RU-C17
Descripción	Se valida que las notificaciones se muestran correctamente en el registro de notificaciones.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas.
Procedimiento	1. Cargar una arquitectura en el simulador. 2. Pulsar el botón ”Info“. 3. Pulsar el botón “Show Notifications”.
Postcondiciones	Las notificaciones que se muestran son las que se han producido.
Evaluación	OK

TABLA 6.40. PRUEBA DE VALIDACIÓN PVA-14

Identificador	PVA-14
Nombre	Copia de seguridad.
Requisitos	RU-C18
Descripción	Se valida que se genera y se carga correctamente la copia de seguridad del código ensamblador introducido y la arquitectura actual.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador. 3. Tener escrito un programa en ensamblador.
Procedimiento	1. Pulsar el botón ”Compile“. 2. Recargar la página web del simulador. 3. Cargar la copia de seguridad.
Postcondiciones	La copia de seguridad se carga correctamente y contiene el código ensamblador introducido y la arquitectura que se estaba utilizando en el momento de realizar la copia de seguridad.
Evaluación	OK

TABLA 6.41. PRUEBA DE VALIDACIÓN PVA-15

Identificador	PVA-15
Nombre	Plataforma.
Requisitos	RU-R01, RU-R02 y RU-R03
Descripción	Se valida que el simulador puede utilizarse en diferentes plataformas.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La simulación se realiza correctamente.
Evaluación	OK

TABLA 6.42. PRUEBA DE VALIDACIÓN PVA-16

Identificador	PVA-16
Nombre	Interfaz de usuario.
Requisitos	RU-R04
Descripción	Se valida que la interfaz de usuario se muestra correctamente en ordenadores y en dispositivos móviles.
Precondiciones	1. Disponer de un dispositivo móvil y un PC. 2. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 3. Tener una arquitectura cargada en el simulador.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La interfaz de usuario se visualiza correctamente en los dispositivos móviles y en los ordenadores.
Evaluación	OK

TABLA 6.43. PRUEBA DE VALIDACIÓN PVA-17

Identificador	PVA-17
Nombre	Tiempo de ejecución de una instrucción.
Requisitos	RU-R05
Descripción	Se valida que el tiempo medio que tarda en ejecutarse una instrucción no es mayor de 0.5 segundos.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa midiendo el tiempo que tarda en ejecutarse cada instrucción.
Postcondiciones	El tiempo medio de ejecución de las instrucciones no sobrepasa los 0.5 segundos.
Evaluación	OK

TABLA 6.44. PRUEBA DE VALIDACIÓN PVA-18

Identificador	PVA-18
Nombre	Conexión a internet.
Requisitos	RU-R06 y RU-R07
Descripción	Se valida que el simulador funciona correctamente sin conexión a internet.
Precondiciones	1. No tener conexión a internet. 2. Tener en el dispositivo el código fuente del simulador. 3. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 4. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Crear un programa ensamblador. 2. Compilar el programa creado. 3. Ejecutar el programa.
Postcondiciones	La simulación se realiza correctamente.
Evaluación	OK

TABLA 6.45. PRUEBA DE VALIDACIÓN PVA-19

Identificador	PVA-19
Nombre	Desarrollo de la herramienta.
Requisitos	RU-R08
Descripción	Se valida que el simulador se ha desarrollado mediante HTML5.
Precondiciones	1. Tener en el dispositivo el código fuente del simulador.
Procedimiento	1. Verificar el código de los ficheros que forman el simulador
Postcondiciones	La herramienta se ha desarrollado mediante HTML5.
Evaluación	OK

TABLA 6.46. PRUEBA DE VALIDACIÓN PVA-20

Identificador	PVA-20
Nombre	Tamaño máximo de la arquitectura.
Requisitos	RU-R09
Descripción	Se valida que no se puede crear una arquitectura de más de 64 bits.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Modificar el campo del tamaño de la arquitectura con un valor mayor de 64 bits.
Postcondiciones	El simulador muestra un error.
Evaluación	OK

TABLA 6.47. PRUEBA DE VALIDACIÓN PVA-21

Identificador	PVA-21
Nombre	Memoria alineada.
Requisitos	RU-R10
Descripción	Se valida que no se puede almacenar datos en memoria si estos no están alineados.
Precondiciones	1. Ejecutar el simulador en un dispositivo con una versión del navegador web igual o superior a las indicadas. 2. Tener cargado en el simulador una arquitectura.
Procedimiento	1. Escribir un programa ensamblador que almacene datos en memoria y que no estén alineados. 2. Compilar el programa.
Postcondiciones	El simulador muestra un error.
Evaluación	OK

A continuación, se muestra una tabla en la que se puede observar que todos los requisitos de usuario han sido verificados.

TABLA 6.48. MATRIZ DE TRAZABILIDAD PRUEBAS DE VALIDACIÓN

Requisitos	PVA-01	PVA-02	PVA-03	PVA-04	PVA-05	PVA-06	PVA-07	PVA-08	PVA-09	PVA-10	PVA-11	PVA-12	PVA-13	PVA-14	PVA-15	PVA-16	PVA-17	PVA-18	PVA-19	PVA-20	PVA-21
RU-C01	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C02	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C03	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C04	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C05	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C06	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C07	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C08	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C09	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C10	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
RU-C11	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-
RU-C12	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RU-C13	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RU-C14	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
RU-C15	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
RU-C16	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
RU-C17	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
RU-C18	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
RU-R01	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RU-R02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RU-R03	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-
RU-R04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
RU-R05	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
RU-R06	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
RU-R07	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
RU-R08	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
RU-R09	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-
RU-R10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓

6.2. Evaluación

Para evaluar el simulador desarrollado, se van a comparar los mensajes de error que este muestra al usuario cuando el código ensamblador que se compila contiene algún error y, además, también se va a comparar la retroalimentación que ofrecen a los usuarios du-

rante la ejecución de un programa, con el fin de verificar que CREATOR es un simulador más didáctico que los ya existentes. Para esta evaluación, se va a utilizar MARS [2], puesto que se trata de uno de los simuladores más utilizados para recrear el funcionamiento de un programa en ensamblador. Para ello, se van a realizar cinco pruebas diferentes donde se podrá observar cómo notifican los errores producidos cada herramienta y cómo es la retroalimentación.

La primera prueba que se realizó fue introducir en un dato de tipo *word* una palabra en vez de un valor numérico. Los resultados obtenidos en ambos simuladores fueron los siguientes:

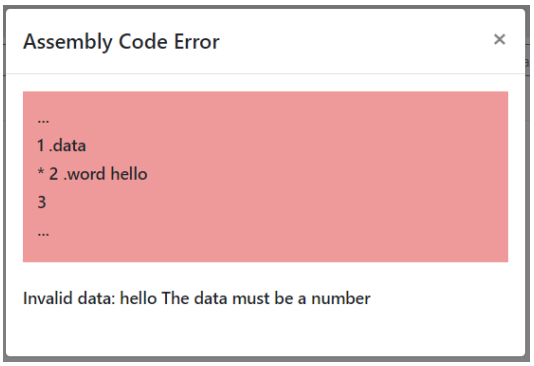


Fig. 6.1. Evaluación 1 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 8: Symbol "hello" not found in symbol table.
Assemble: operation completed with errors.
```

Fig. 6.2. Evaluación 1 MARS

Como se puede observar, CREATOR indica el error producido con más precisión que MARS, lo que permite a los usuarios comprender mejor el error que se ha producido y así subsanarlo con mayor rapidez.

La siguiente prueba realizada consiste en introducir una instrucción que no está definida en la herramienta. A continuación, se muestran los resultados obtenidos:

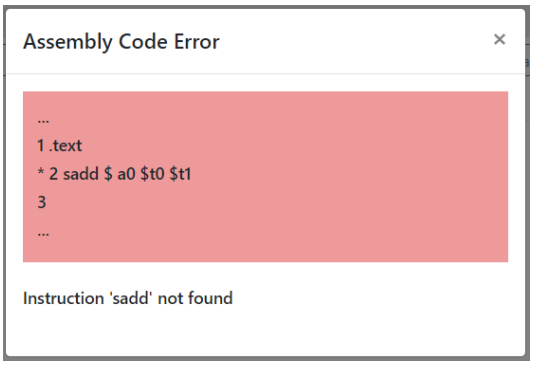


Fig. 6.3. Evaluación 2 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 2: "sadd" is not a recognized operator
Assemble: operation completed with errors.
```

Fig. 6.4. Evaluación 2 MARS

Al igual que en la comparación anterior, el resultado de CREATOR es más intuitivo que el mostrado por MARS.

La tercera prueba que se va a realizar consiste en introducir el nombre de un registro que no está definido en el simulador. Los resultados obtenidos son:

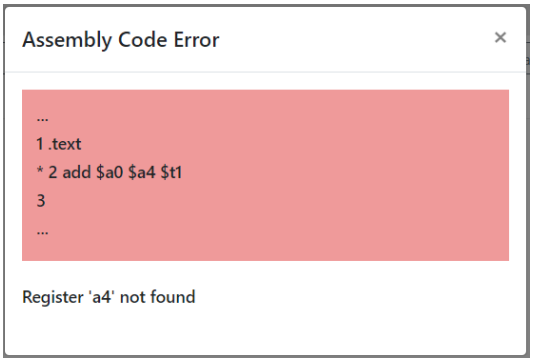


Fig. 6.5. Evaluación 3 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 10: "$a4": operand is of incorrect type
Assemble: operation completed with errors.
```

Fig. 6.6. Evaluación 3 MARS

En esta prueba se sigue la misma dinámica que en los dos casos anteriores.

La cuarta prueba consiste en introducir una directiva que no está definida en la herramienta. El resultado obtenido en ambas herramientas es:

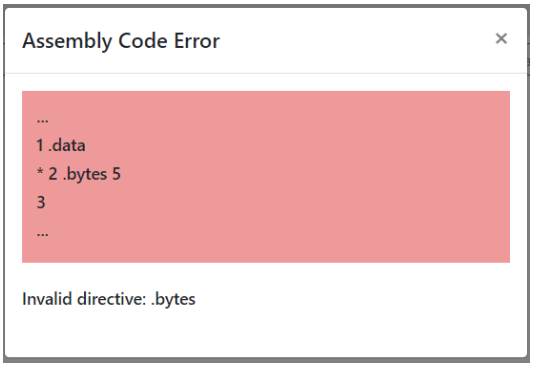


Fig. 6.7. Evaluación 4 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Warning in D:\Users\Diego\Downloads\mips1.asm line 2 column 2: MARS does not recognize the .bytes directive.  Ignored.
Assemble: operation completed successfully.
```

Fig. 6.8. Evaluación 4 MARS

Como se puede observar, CREATOR notifica el error de forma detallada al usuario. MARS, sin embargo, compila el programa completo ignorando la directiva introducida. Esto supone un problema para los usuarios ya que están introduciendo un código ensamblador erróneo, pero el simulador no notifica correctamente el fallo realizado pudiéndose pasar por alto este error.

La última comparación que se ha realizado entre estos dos simuladores consiste en analizar cómo indica cada simulador la instrucción que se ha ejecutado y la que se va a ejecutar en el siguiente paso. Los resultados obtenidos son:

Break	Address	Label	User Instructions	Loaded Instructions
	0x0	main	beq $5 $a1 b	beq $5 $a1 0x10
	0x4	c	addi $a0 $a1 10	addi $a0 $a1 10
	0x8		addi $a0 $a1 100	addi $a0 $a1 100
	0xc		mul $4 $a0 $a0	mul $4 $a0 $a0
	0x10	b	addi $a0 $a1 15265	addi $a0 $a1 15265
	0x14		addi $a0 $a1 105525452	lui $at 0x064A
	0x18			ori $at $at 0x30CC
	0x1c			add $a0 $a1 $at
	0x20		move $4 $0	add $4 $r0 $0

Fig. 6.9. Evaluación 5 CREATOR

Bkpt	Address	Code	Basic	Source
	0x00400000	0x10a50003	beq $5,$5,0x00000003	5: beq $5 $a1 b
	0x00400004	0x20a4000a	addi $4,$5,0x0000000a	6: c: addi $a0 $a1 10 #oihewfolfjoijioerg
	0x00400008	0x20a40064	addi $4,$5,0x00000064	7: addi $a0 $a1 100
	0x0040000c	0x70842002	mul $4,$4,$4	8: mul $4 $a0 $a0
	0x00400010	0x20a43ba1	addi $4,$5,0x00003ba1	9: b: addi $a0 $a1 15265
	0x00400014	0x3c01064a	lui $1,0x0000064a	10: addi $a0 $a1 105525452
	0x00400018	0x342130cc	ori $1,$1,0x000030cc	
	0x0040001c	0x00a12020	add $4,$5,$1	
	0x00400020	0x00002021	addu $4,$0,$0	11: move $4 $0

Fig. 6.10. Evaluación 5 MARS

Como se puede observar en las ilustraciones anteriores, CREATOR indica ,en tiempo de ejecución, la instrucción que se acaba de ejecutar (azul) y la próxima instrucción que se va a ejecutar (verde), mientras que MARS solo muestra la próxima instrucción que se va a ejecutar.

Después de realizar estas cinco pruebas, se ha podido comprobar que la herramienta desarrollada es más intuitiva y didáctica que otras ya existentes y que son muy utilizadas, ya que los errores que el usuario comete al escribir un programa ensamblador son indicados de una forma más precisa, lo que es de gran ayuda cuando se está aprendiendo. Además, CREATOR, cuando está ejecutando un programa, muestra tanto la última instrucción ejecutada como la siguiente que se va a ejecutar. Esto es un aspecto importante, sobre todo en el caso de que el programa contenga saltos, ya que permite verlos rápidamente y así comprender de forma más eficiente el funcionamiento del programa e incluso depurar el programa frente a posibles errores de implementación que se hayan podido producir.

Capítulo 7

Planificación y presupuesto

En este capítulo se va a presentar la planificación que se ha seguido para realizar este proyecto (Sección 7.1). Después, se detallarán los costes del proyecto (Sección 7.2) y el entorno socio-económico (Sección 7.3).

7.1. Planificación

En esta sección se va a detallar la planificación que se utiliza para realizar el proyecto. En primer lugar, se va a realizar un análisis de distintas metodologías de desarrollo para, después, explicar el método que se ha elegido para este caso y el porqué. Después, se muestra, de forma detallada, las distintas fases por las que ha pasa el proyecto y la duración de cada una mediante un diagrama de Gantt.

7.1.1. Metodología

En la actualidad, existen diferentes metodologías de desarrollo, por ello, a continuación, se van a analizar tres metodologías y así conocer las características principales de cada una.

- **Modelo en cascada [33]:** este modelo está basado en un diseño secuencial, es decir, cada una de las fases (análisis, diseño, implementación, pruebas y evaluación) de las que consta el proyecto se comienza a realizar cuando se ha terminado la fase an-

terior. Esta metodología se suele utilizar en proyectos sencillos y de corta duración, ya que no permite realizar iteraciones en el desarrollo del software.

- **Modelo de prototipos [34]:** este modelo se fundamenta en la realización de un prototipo inicial de forma rápida. Este modelo tiene el inconveniente de que en algunos proyectos no es viable realizar un prototipo inicial con rapidez debido a la complejidad de este.

- **Modelo en espiral [35]:** este modelo se apoya en la fragmentación del desarrollo del proyecto en varias iteraciones, ya que es un modelo iterativo. Este modelo es costoso porque requiere realizar varias iteraciones de todas las fases del proyecto.

Después de analizar las diferentes metodologías se ha decidido utilizar el modelo en espiral que, aunque siendo más costoso que las otras metodologías debido a la complejidad del proyecto, es más conveniente realizarlo en varias iteraciones. Además, el modelo de prototipos no era viable debido a que no se podía realizar un prototipo en poco tiempo por la complejidad del proyecto.

7.1.2. Ciclo de vida

El ciclo de vida de este proyecto sigue el Modelo de ciclo de vida en Espiral [35]. Este modelo está formado por cuatro etapas, las cuales son realizadas en cada una de las iteraciones que se realizan a lo largo del desarrollo del proyecto. Las etapas son:

- **Planificación:** se obtienen los requisitos de usuario, y tras analizarlos se deciden los objetivos de la iteración que se va a realizar.

- **Análisis:** se analizan los requisitos obtenidos para identificar los riesgos que se pueden llegar a dar y las diferentes alternativas que existen.

- **Desarrollo y pruebas:** en esta fase se implementa el código y también se realizan casos de prueba para verificar el correcto funcionamiento del software.

- **Evaluación:** el cliente evalúa el sistema para aportar comentarios. Si se obtiene la aprobación del cliente se pasará a la siguiente iteración, en caso contrario se tendrá

que llevar a cabo los comentarios que el cliente ha transmitido antes de comenzar la siguiente iteración.

Este proyecto estará formado por cinco iteraciones. Estas iteraciones son:

- **Interfaz de usuario:** en esta iteración se desarrolla la interfaz de usuario simple, que se utilizará para tener una visión general de la organización de los diferentes elementos que la forman y poder incorporar después las funcionalidades del simulador.
- **Editor de arquitectura:** en esta iteración se realiza el módulo encargado de la definición de una arquitectura.
- **Kernel de simulación:** se realiza el módulo encargado de ejecutar las instrucciones ensamblador introducidas por el usuario.
- **Compilador:** se desarrolla el compilador del simulador. Este se encarga de verificar que las instrucciones introducidas por el usuario son correctas y almacenarlas, en este caso, en la memoria del simulador para que puedan ser ejecutadas y, en el caso de que contengan algún fallo, se encargará de notificar al usuario del error.
- **Bibliotecas:** en esta iteración se realiza la funcionalidad que permite al usuario cargar y guardar una biblioteca de funciones que se podrá ejecutar junto con el código introducido.

7.1.3. Tiempo estimado

El diagrama de Gantt que se muestra en la Figura 7.1 detalla las tareas que se han realizado durante el desarrollo del proyecto, además del tiempo que se dedica a cada una de ellas. El proyecto se comenzó el 10 de septiembre de 2018 y se finalizó el 10 de junio de 2019. Durante este tiempo se ha trabajado en el proyecto veinte horas semanales, lo que hace un total de 780 horas que corresponde a 39 semanas.

En el diagrama de Gantt se detallan todas las iteraciones realizadas y en cada una de ellas se indican las etapas que se realizan y su duración. Las iteraciones y las fases que se muestran son las explicadas en la Sección 7.1.2, a excepción de una fase extra llamada Documentación en la que se realizará la presente documentación.

Nombre de la tarea	Inicio	Final	Duración
Iteración 1	**10-sep.-18**	**16-oct.-18**	**36**
Planificación	10-sep.-18	13-sep.-18	3
Análisis	13-sep.-18	20-sep.-18	7
Desarrollo y Pruebas	20-sep.-18	7-oct.-18	17
Evaluación	7-oct.-18	12-oct.-18	5
Documentación	12-oct.-18	16-oct.-18	4
Iteración 2	**16-oct.-18**	**14-dic.-18**	**59**
Planificación	16-oct.-18	21-oct.-18	5
Análisis	21-oct.-18	29-oct.-18	8
Desarrollo y Pruebas	29-oct.-18	1-dic.-18	33
Evaluación	1-dic.-18	8-dic.-18	7
Documentación	8-dic.-18	14-dic.-18	6
Iteración 3	**14-dic.-18**	**21-feb.-19**	**69**
Planificación	14-dic.-18	19-dic.-18	5
Análisis	19-dic.-18	29-dic.-18	10
Desarrollo y Pruebas	29-dic.-18	3-feb.-19	36
Evaluación	3-feb.-19	13-feb.-19	10
Documentación	13-feb.-19	21-feb.-19	8
Iteración 4	**21-feb.-19**	**9-may.-19**	**77**
Planificación	21-feb.-19	26-feb.-19	5
Análisis	26-feb.-19	10-mar.-19	12
Desarrollo y Pruebas	10-mar.-19	19-abr.-19	40
Evaluación	19-abr.-19	2-may.-19	13
Documentación	2-may.-19	9-may.-19	7
Iteración 5	**9-may.-19**	**10-jun.-19**	**32**
Planificación	9-may.-19	12-may.-19	3
Análisis	12-may.-19	17-may.-19	5
Desarrollo y Pruebas	17-may.-19	28-may.-19	11
Evaluación	28-may.-19	31-may.-19	3
Documentación	31-may.-19	10-jun.-19	10

Fig. 7.1. Diagrama de Gantt

7.2. Presupuesto

En este apartado se va a detallar el presupuesto del proyecto. En primer lugar, se detallarán los coste del proyecto y, después, se muestra la oferta presentada al cliente.

7.2.1. Coste del proyecto

En la siguiente tabla se presenta un resumen de las principales características del proyecto, así como, el presupuesto final.

TABLA 7.1. RESUMEN DEL PROYECTO

Titulo	Diseño y desarrollo de un simulador genérico para programación en ensamblador
Autor	Diego Camarmas Alonso
Departamento	Departamento de Informática
Fecha de inicio	10 de septiembre de 2018
Fecha de finalización	10 de junio de 2019
Duración	9 meses
Presupuesto total	71.076,21 €

Costes directos

En este sección se muestran los costes directos del proyecto. En la Tabla 7.2 se detallan los costes derivados del personal que ha trabajado en el desarrollo de este proyecto. Los roles que han ejercido el tutor y el estudiante son:

- **Tutor:** Jefe de proyecto
- **Estudiante:** Analista, Desarrollador y Probador

TABLA 7.2. COSTES EN RECURSOS HUMANOS

Categoría	Coste por hora (€)	Horas	Total (€)
Jefe de proyecto	60	59	3.540
Analista	40	308	12.320
Desarrollador	35	332	11.620
Probador	35	216	7.560
Total			**35.040**

En la Tabla 7.3 se detallan los costes directos asociados a la compra y uso de diferentes equipos para realizar este proyecto. El coste que supone el uso de cada equipo en este proyecto se calculará respecto al tiempo que este es usado en el proyecto. Para ello se utiliza la siguiente formula:

$$c = \frac{t \cdot C \cdot p}{a} \tag{7.1}$$

Donde:

- **c:** coste amortizado.
- **t:** tiempo que se utiliza el equipamiento para el realizar el proyecto.
- **C:** coste del equipamiento.
- **p:** porcentaje de tiempo que se ha utilizado el equipamiento durante el proyecto.
- **a:** periodo de amortización del equipamiento.

TABLA 7.3. COSTES EN EQUIPAMIENTO

Concepto	Coste, C (€)	Dedicación, p (%)	Dedicación, t (meses)	Amortización, a (meses)	Coste amortizado, c (€)
PC sobremesa	706,10	50	9	36	88,26
Portátil	568,35	100	9	36	142,09
Smartphone	469	25	9	24	43,97
Tablet	269	25	9	36	16,81
Impresora	100,35	5	9	60	0,75
Total					**291,88**

A continuación, se van a detallar las características de cada uno de los equipos utilizados en el proyecto:

- **Ordenador de sobremesa:** PC genérico, i5-4690, 8GB RAM, 1TB HDD, 128GB SSD, GTX 750.
- **Portátil:** Lenovo G50-80, i5-5200U, 8GB RAM, 500GB SSD.
- **Smartphone:** Huawei p20.
- **Tablet:** Samsung Galaxy Tab A 10.1" 2019 64GB Wifi.
- **Impresora:** HP LaserJet Pro MFP M130a.

Otros costes que se deben tener en cuenta, además, de los anteriores son los costes del material fungible, que se muestran en la Tabla 7.4. Estos costes están formados por material de oficina y un tóner para la impresora. En el material de oficina se incluyen: bolígrafos, lapiceros, subrayadores, típex y folios.

TABLA 7.4. COSTES DE MATERIAL FUNGIBLE

Concepto	Coste (€)
Material de oficina	86,54
Tóner (x1)	53,00
Total	**139,54**

Resumen de costes

Una vez calculados los costes directos se va a mostrar un resumen de estos (Tabla 7.5), además, también se mostrarán los costes indirectos asociados al proyecto que serán del 20 % de los costes directos. Los costes indirectos incluyen el coste del agua, luz, acceso a internet, etc.

TABLA 7.5. RESUMEN DE LOS COSTES

Concepto	Coste (€)
Recursos humanos	35.040
Equipos	291,88
Material fungible	139,54
Costes indirectos	7.094,28
Total	**42.565,70**

7.2.2. Oferta del proyecto

A continuación, se muestra la Tabla 7.6 en la que se detalla la oferta propuesta. En la oferta se incluyen unos riesgos estimados del 15 %, unos beneficios estimados del 20 % y, por último, el Impuesto de Valor Añadido (IVA) que en España es del 21 % [36]. Por lo tanto, una vez aplicados todos estos conceptos, el coste final de este proyecto es de **71.076,21 €(Setenta y un mil setenta y seis euros con veintiún céntimos)**.

TABLA 7.6. RESUMEN DE LOS COSTES

Concepto	Incremento (%)	Valor parcial (€)	Coste agregado (€)
Coste del proyecto	-	42.565,70	42.565,70
Riesgos	15	6.384,86	48.950,56
Beneficio	20	9.790,11	58.740,67
IVA	21	12.335,54	71.076,21
Total			**71.076,21**

7.3. Entorno socio-económico

Este proyecto otorga beneficios en la enseñanza de las asignaturas de Estructura y Arquitectura de Computadores haciendo que el esfuerzo que los alumnos tienen que realizar para aprender estos conocimientos sea menor. Esto se debe a:

- **Disponibilidad de simular distintas arquitecturas en una misma herramien-**

ta. Esto permite que los usuarios necesiten menos tiempo para aprender a utilizar la herramienta, al no tener que utilizar una herramienta distinta para cada tipo de arquitectura.

- **Múltiples casos prácticos**. Debido a que el simulador es genérico los usuarios pueden realizar ejecuciones de programas ensamblador con diferentes tipos de instrucciones, registros o directivas.

- **Se puede utilizar en cualquier momento**. Como se trata de una aplicación multiplataforma se puede utilizar en un ordenador personal, *smartphone* o *tablet* y, además, no necesita conexión a internet para realizar su función, por lo tanto, esto permite que se pueda utilizar en múltiples lugares y situaciones.

- **Código abierto**. Esta herramienta se encuentra bajo la Licencia Pública General Menor de GNU (LGPL), ya que se pretende que este simulador no solo sea una herramienta para los alumnos de la Universidad Carlos III de Madrid, sino que pueda ser una herramienta de libre uso para toda la comunidad educativa, es decir, que pueda ser utilizada por otras universidades para mejorar la docencia de las asignaturas de Arquitectura y Estructura de Computadores, sin tener que realizar un desembolso económico para poder acceder a ella.

- **Idioma**. El simulador está escrito en inglés para que la difusión de la herramienta pueda ser lo más amplia posible.

Capítulo 8

Conclusiones y trabajos futuros

En este capítulo se van a presentar unas conclusiones del proyecto (Sección 8.1), donde se estudiarán los objetivos indicados en el Capitulo 1, a continuación, se indicarán las asignaturas estudiadas a lo largo del grado que están relacionadas con este proyecto (Sección 8.2) y, para finalizar, se indican una serie de trabajos futuros (Sección 8.3).

8.1. Conclusiones

En este punto se va a estudiar si se han cumplido o no los objetivos propuestos en la Sección 1.2 del Capítulo 1 Introducción.

El primer objetivo que se propuso consistía en que el simulador permitiría definir diferentes juegos de instrucciones, definir la arquitectura y que el simulador fuera lo suficiente genérico para poder definir cualquier juego de instrucciones. Para cumplir este objetivo se tuvo que realizar un estudio de los atributos que forman las instrucciones, pseudoinstrucciones y registros de diferentes arquitecturas para seleccionar aquellos que son comunes y esenciales para una correcta definición del componente. Además, también se tuvo que estudiar la organización de la estructura de datos que almacena todos los componentes que forman una arquitectura para que fuera compacta, eficiente en las busquedas y fácilmente editable, debido a que esto es una característica muy importante de la herramienta desarrollada. Por lo tanto, como se puede ver en la presente memoria este objetivo se ha conseguido llevar a cabo.

El segundo objetivo consistía en que el simulador debía permitir escribir programas en lenguaje ensamblador utilizando el juego de instrucciones disponible. Para llevar a cabo este objetivo se tuvo que diseñar e implementar un compilador de lenguaje ensamblador. Este compilador se encarga de verificar que el código escrito corresponde con el juego de instrucciones que se ha cargado en el simulador y, en el caso de que esto no sea así, notificar al usuario del error producido. Como se puede apreciar en los capítulos anteriores este objetivo se ha cumplido.

El tercer objetivo indica que la herramienta debe ser capaz de compilar programas escritos en lenguaje ensamblador. Para ello se diseñó e implementó un compilador, al igual que para el objetivo anterior. Por lo tanto, el objetivo se ha cumplido como se puede ver a lo largo de la presente memoria.

El siguiente objetivo presentado indica que la herramienta permitiría ejecutar y depurar programas en ensamblador. Para poder cumplir con este objetivo se realizó un módulo que se encarga de esta funcionalidad. Para realizar la ejecución se tiene que obtener de la memoria del simulador las instrucciones y datos y, mediante la definición de la instrucción, se realizan las operaciones correspondientes. Para permitir la depuración de la ejecución del programa se crearon varias vistas en la interfaz de usuario que permiten ver las estadísticas de la ejecución, los valores almacenados en la memoria principal y en los registros, además, en este último caso, los valores almacenados se muestran en distintos formatos. Finalmente este objetivo también se ha podido cumplir.

El quinto objetivo señala que se debía poder visualizar el estado del computador después de ejecutar cada una de las instrucciones del programa. Para poder llevar a cabo este objetivo se ofrecen dos tipos de ejecución, instrucción a instrucción o el programa completo de seguido, en ambos casos se puede visualizar en tiempo de ejecución el valor almacenado en los registros y memoria principal, ya que estos valores son actualizados después de ejecutar cada instrucción. Como se puede ver en la presente memoria, este objetivo se cumple.

El último objetivo presentado consistía en que el simulador debía permitir visualizar y editar los diferentes atributos de los componentes que forman la arquitectura. Para ello se crearon varias vistas en la interfaz de usuario que mostraban todos los componentes y sus atributos y, para poder editar los atributos, se crearon diferentes formularios que

permitían modificar los valores de estos. Este objetivo también se ha podido realizar como se ha podido observar a lo largo de esta memoria.

8.2. Contribuciones

Para poder realizar este proyecto se han utilizado conocimientos adquiridos en diferentes asignaturas del Grado en Ingeniería Informática. Estas asignaturas son:

- **Tecnología de Computadores** (asignatura obligatoria impartida en el primer curso). En esta asignatura se estudian los componentes hardware y la lógica binaria que utilizan estos componentes.
- **Estructura de Computadores** (asignatura obligatoria impartida en el segundo curso). En esta asignatura se estudia la estructura que tiene un computador, el lenguaje ensamblador MIPS32 y microprogramación.
- **Sistemas Operativos** (asignatura obligatoria impartida en el segundo curso). En esta asignatura se estudia el funcionamiento de un Sistema Operativo.
- **Arquitectura de Computadores** (asignatura obligatoria impartida en el tercer curso). En esta asignatura se imparten las bases de una arquitectura de un computador.
- **Interfaces de Usuario** (asignatura obligatoria impartida en el tercer curso). En esta asignatura se estudia cómo se deben organizar los componentes de una interfaz de usuario y también cómo se realiza una página web que se ejecuta en el lado cliente.
- **Diseño de Sistemas Operativos** (asignatura obligatoria impartida en el tercer curso). En esta asignatura se estudia cómo están diseñados los distintos módulos que forman un Sistema Operativo.
- **Dirección de Proyectos de Desarrollo de Software** (asignatura obligatoria impartida en el tercer curso). En esta asignatura se estudia cómo se debe realizar la dirección y gestión de los proyectos de desarrollo de software.
- **Computación Ubicua** (asignatura optativa impartida en el cuarto curso). En esta asignatura se estudia cómo se realiza una aplicación multiplataforma. Además, también se estudia cómo se crea una aplicación para el Sistema Operativo Android.

8.3. Trabajos futuros

Una vez finalizado este proyecto, se van a presentar una serie de trabajos futuros que servirán para dotar a esta herramienta de más funcionalidades. Estos trabajos son:

- Ampliar el simulador con otros lenguajes ensamblador como ARM o ensamblador para arquitectura Intel.
- Incorporar un simulador de caché.
- Ampliar el simulador con funciones a utilizar por los profesores para ayudar a la corrección de las prácticas.

Anexo A

Manual de usuario

En este anexo se va a presentar el manual de usuario de CREATOR. Este manual estará dividido en distintas secciones. En la primera sección se van a detallar los pasos que se deben seguir para ejecutar un código ensamblador en el simulador (Sección A). Después, se explicarán los pasos que se deben realizar para cargar una arquitectura en el simulador (Sección B) y, a continuación, se describen las acciones que se pueden realizar para gestionar el código ensamblador (Sección C). Por último, se detallarán las acciones que se pueden realizar en la pantalla principal para realizar una simulación y, también, se explica la información que se muestra en esta pantalla (Sección D).

A. Pasos para ejecutar en CREATOR

1. Cargar una arquitectura en el simulador.
2. Escribir un programa ensamblador que hace uso de las instrucciones definidas en la arquitectura cargada y compilarlo.
3. Ejecutar el programa cargado.

B. Carga de la arquitectura a utilizar

Para cargar una arquitectura en el simulador existen dos métodos, el primer método carga en el simulador una arquitectura que la propia herramienta proporciona y el segundo método consiste en cargar una arquitectura que no proporciona el simulador.

Para cargar una arquitectura proporcionada por el simulador, el único paso que tiene que realizar el usuario es seleccionar en el menú la arquitectura deseada como se muestra en la siguiente imagen.

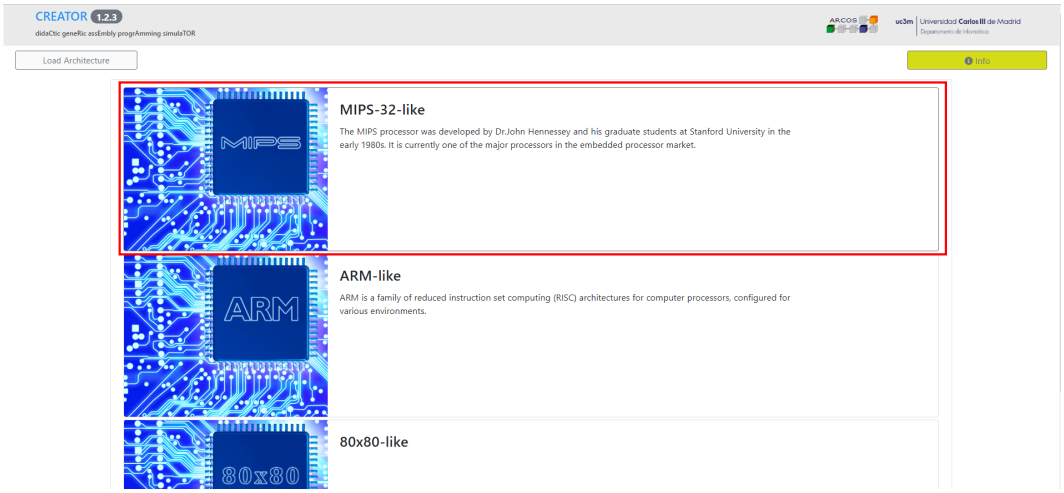


Fig. B.1. Carga de una arquitectura proporcionada

Para cargar una arquitectura que no proporciona el simulador hay que realizar los siguientes pasos:

1. Pulsar el botón “Load Architecture”.
2. Se completan los campos del formulario excepto, si se desea, el campo descripción ya que es opcional. Este paso se puede observar en la imagen B.2
3. Se selecciona en el menú la arquitectura que se ha cargado tal y como se muestra en la imagen B.3.

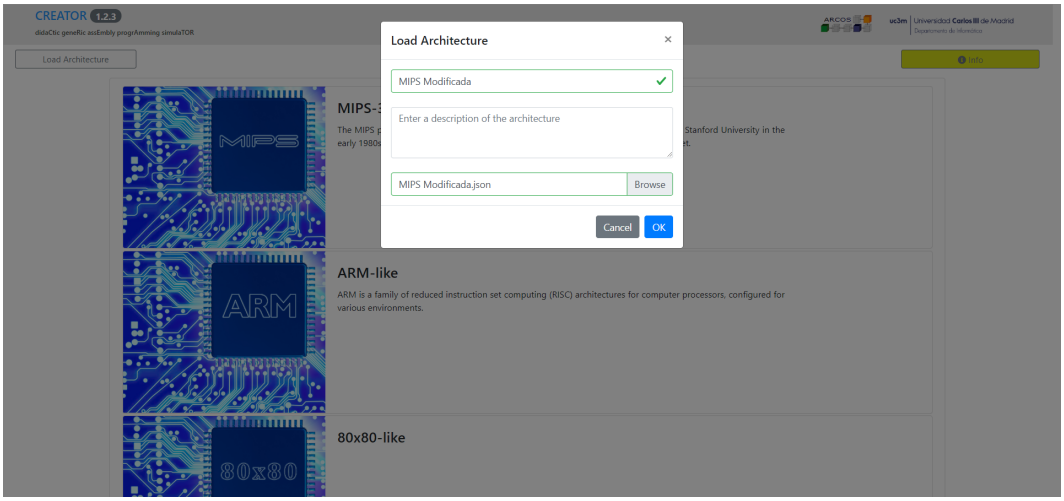


Fig. B.2. Formulario de carga de una nueva arquitectura

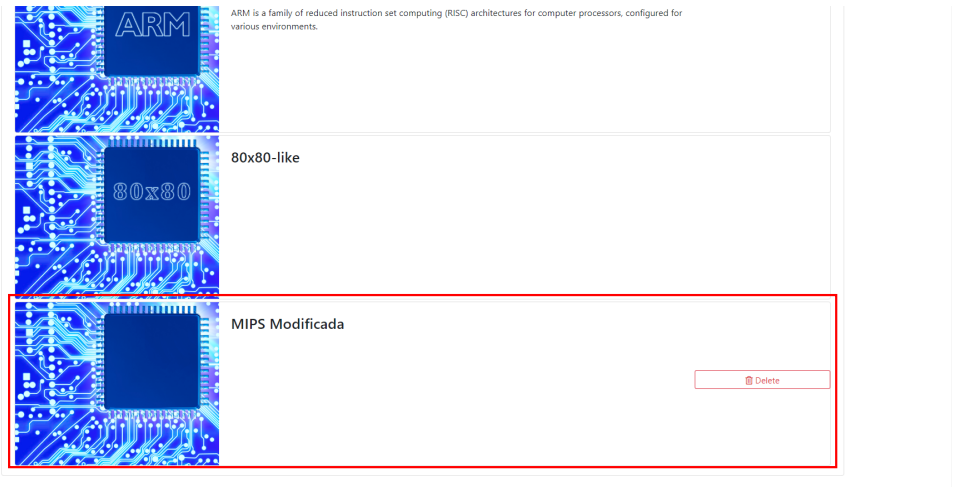


Fig. B.3. Carga de la nueva arquitectura

C. Gestión del código ensamblador

Para gestionar el código ensamblador se debe pulsar el botón “Assembly” que se encuentra en la parte superior de la pantalla, como se puede ver en la siguiente figura.

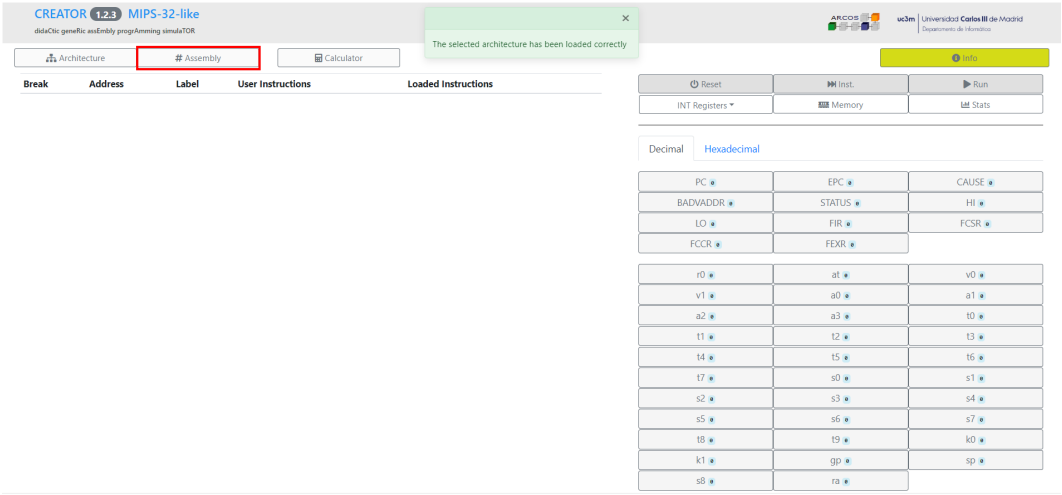


Fig. C.1. Selección del botón “Assembly”

En la pantalla de gestión del código ensamblador, el usuario puede realizar diversas acciones. Estas acciones y pasos que se deben realizar para llevarlas a cabo son:

- Escribir un programa en ensamblador en el área de texto.
- Crear un nuevo programa ensamblador. File → New

- Cargar un programa que se encuentra en el dispositivo utilizado. File → Load
- Guardar el programa escrito en el simulador en el dispositivo que se está utilizando. File → Save
- Cargar un programa de ejemplo que proporciona el simulador. File → Examples → Seleccionar el ejemplo deseado
- Compilar el programa ensamblador. Para ello se pulsa el botón “Compile/Linked”.
- Crear una biblioteca. Library → Create
- Cargar una biblioteca. Library → Load Library
- Eliminar la biblioteca cargada. Library → Remove

A continuación, se muestra una imagen de la pantalla “Assembly” donde se pueden realizar todas las acciones anteriores.

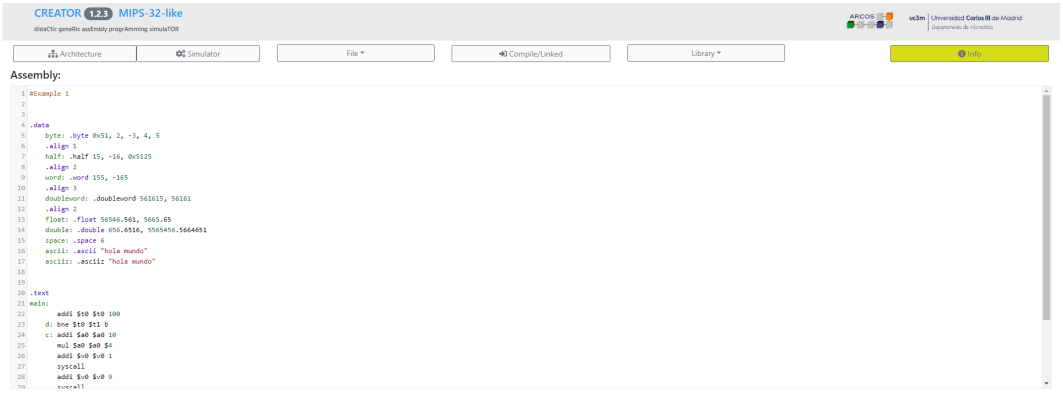


Fig. C.2. Pantalla Assembly

D. Simulación

En la pantalla principal se podrá ejecutar el código ensamblador compilado y ver el estado del simulador después de ejecutar cada una de las instrucciones o al finalizar el programa.

En esta pantalla se puede visualizar:

- El código ensamblador que se va a ejecutar. Parte izquierda de la figura D.1
- Los registros que tiene la arquitectura cargada y el valor que almacena. Parte derecha de la figura D.1
- Una consola. Parte inferior de la figura D.1
- La memoria del simulador. Esta estará dividida en segmento de texto, datos y pila. Parte derecha de la figura D.2
- Estadísticas de ejecución. Parte derecha de la figura D.3
- Una calculadora de coma flotante de 32 bits y 64 bits. Figura D.4

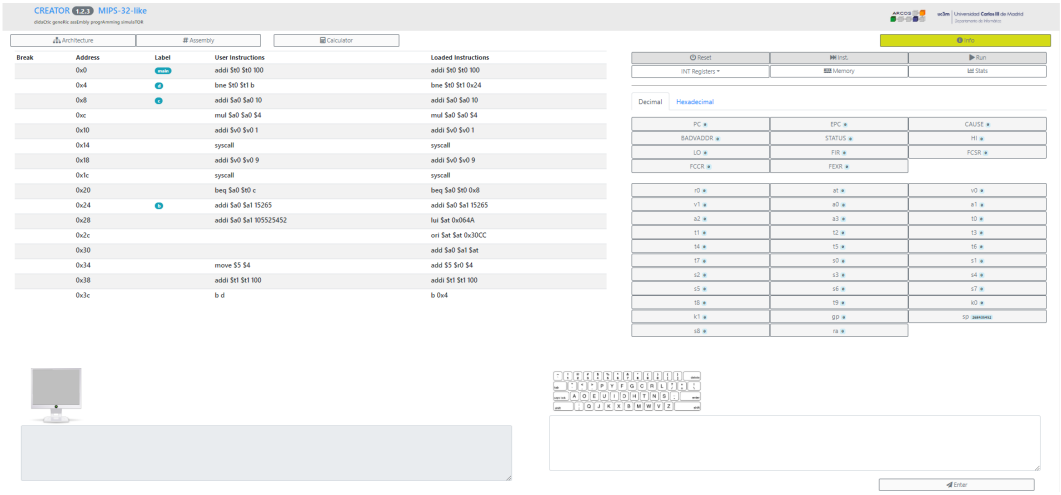


Fig. D.1. Pantalla principal

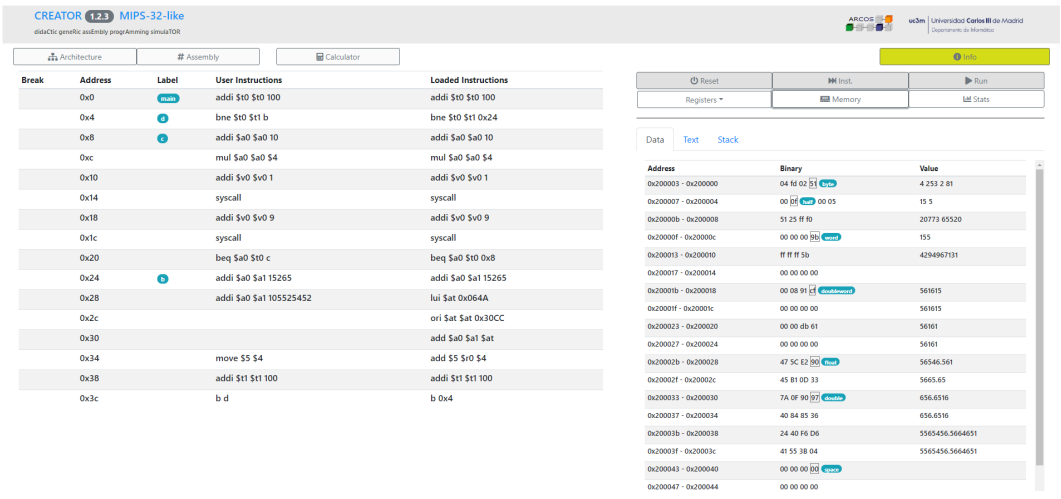


Fig. D.2. Pantalla principal: visualización de la memoria

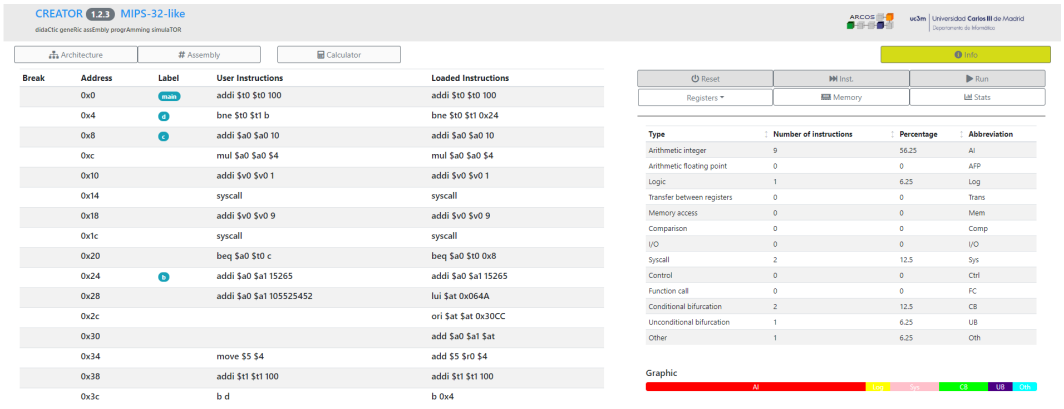


Fig. D.3. Pantalla principal: visualización de las estadísticas

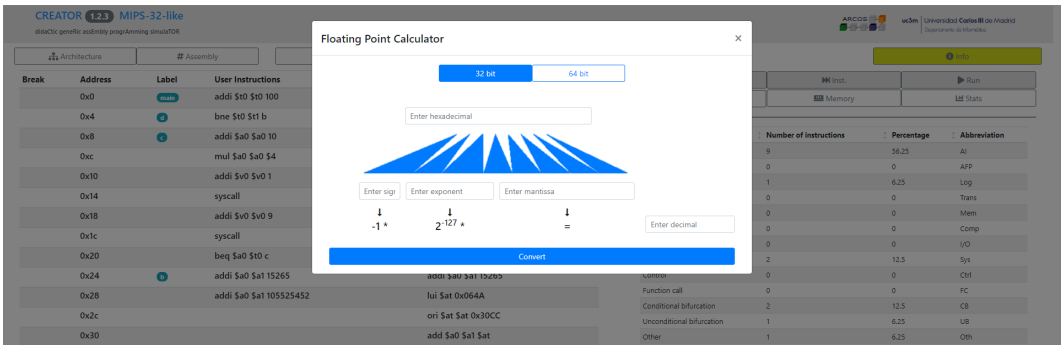


Fig. D.4. Pantalla principal: visualización de la calculadora

Las acciones que se pueden realizar en esta pantalla son:

- Ejecutar el código ensamblador instrucción a instrucción. Pulsar el botón “Inst”.
- Ejecutar el programa completo o hasta un punto de ruptura. Pulsar el botón “Run”
- Reiniciar el simulador. Pulsar el botón “Reset”
- Insertar un *breakpoint*. Seleccionar la instrucción en la que se quiere introducir.
- Mostrar una calculadora de coma flotante. Pulsar el botón “Calculator”

A continuación, se muestra una imagen donde se puede observar un programa durante su ejecución, la instrucción que está coloreada de azul es la última instrucción ejecutada y la que está coloreada de verde es la instrucción que se va a ejecutar en el próximo paso. El icono “Stop” indica que en esa instrucción se ha insertado un *breakpoint*.

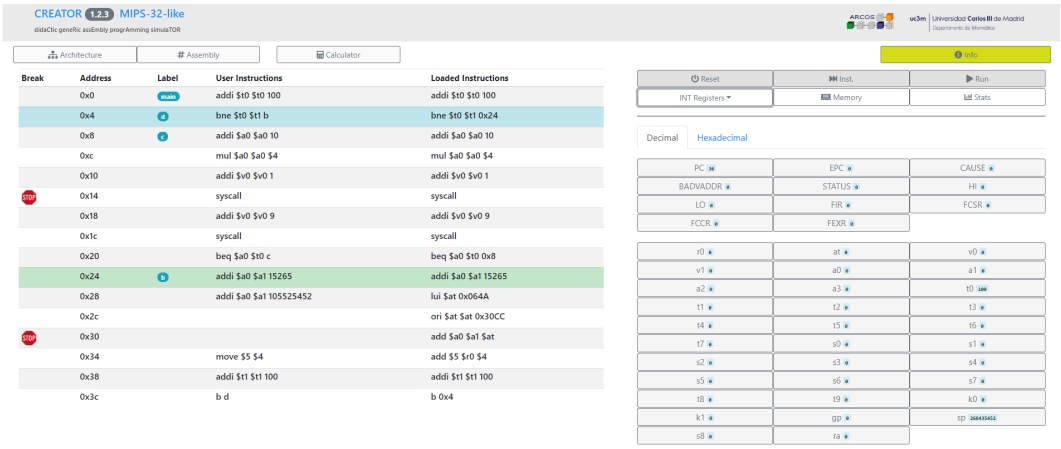


Fig. D.5. Pantalla principal: visualización de una ejecución

Appendix B

Extended abstract

In this Appendix we are going to make a long summary about this project. First, there will be an introduction (Section A) in which the motivations and objectives of this project will be detailed. Next, the design of the simulator will be described (Section B), indicating the final solution chosen and the architecture of the simulator. Then, an evaluation of the developed simulator will be carried out against another simulator (Section C). Finally, some conclusions will be presented (Section D).

A. Introduction

A.1. Motivation

In Computer Engineering, a fundamental piece is to know how is the architecture of a computer since it allows to understand the operation of this at low level. It is for this reason that its teaching must be as complete as possible, but at the same time, it should not suppose great challenges to the students at the time of realizing the practical exercises obtaining this way a much greater understanding of the operation of a computer.

One of the main challenges faced by teachers who teach the subjects of Computer Structure and Architecture when preparing the practical classes is to choose the right tool that students will use, since this tool has to simulate the operation of a computer as similar as possible to what is taught in the theoretical classes.

Currently, there are different tools that can simulate the internal functioning of a

computer, but these are too complex to use, not very intuitive and are not accessible by mobile devices (Smartphones or Tablets), since most are developed for computers, and are usually designed to simulate a single architecture.

This is why we have decided to design and develop a generic simulator for programming in assembly and called CREATOR (didaCtic geneRic assEmbly prograMming simulaTOR). It was thought that this simulator should be generic to be able to simulate different types of architectures with the same tool and that allows students to be more familiar with the operation of this tool and, therefore, perform tasks faster and easier. This feature also allows new architectures to be added at a later date, which gives it added value, as the world of computing advances at great speed and teaching must go hand in hand. For all these reasons, the development of this project has begun, which is being carried out as part of a teaching innovation project at the Carlos III University of Madrid during the academic year 2018 - 2019 entitled "Improvement of teaching resources for practical work in the subject Structure of Computers", which has been coordinated by Félix García Carballeira and which will be used in the next few years in this subject.

This tool, besides being able to simulate different architectures, also allows students to edit some of the characteristics of the architecture in which they are working, such as creating, modifying or deleting registers, assembly instructions or pseudointructions.

As mentioned above, most of the tools currently available for this purpose are developed for computers, but now there are more and more mobile devices (Smartphones or tablets) so it was decided that this tool could also be used in mobile devices and thus not limit their use.

Therefore, we propose to develop a simulator that is: simple, intuitive, generic and multiplatform and in which it is possible to understand the operation of a computer at low level thanks to the creation and execution of programs in different assembly languages and the great freedom that users have to edit the architecture on which they are working.

A.2. **Objectives**

In this project, the main objective is to make a generic simulator to program in assembler and that allows to execute assembler programs on different types of architectures,

at the same time to edit them. This allows students to understand the operation of different types of architectures in a simpler and more interactive way, as well as learn to program in assembly language.

In addition to the main objective, the simulator must allow:

- Define different sets of instructions. The user will be able to define the architecture, components and set of instructions. The simulator must be generic enough to define any set of instructions.
- Write programs in assembly language using the available instruction sets.
- Execute programs in assembler and debug their execution.
- To visualize the state of the computer after executing each one of the instructions that conform a program.
- Visualize and edit the different attributes of the components that make up the architecture.

B. Design

B.1. Final solution

In order to design the tool, it was necessary to take into account the user requirements that indicate the capabilities and limitations of the system. Therefore, after analyzing the requirements and the different alternatives that exist to make a tool of this type, it has been decided that this will be developed as a web application formed solely by the client side, since this type of applications can be run on different devices without the need to have a server and regardless of the operating system used, because it is only dependent on the web browser in which the application is running. All this allows the application to be used at any time and place on a device, as it does not require a previous installation of the application.

In order to carry out the implementation, HTML5 will be used, which includes HTML, CSS and JavaScript, allowing the application to be used in computers, tablets,

smartphones and other types of devices that allow the use of web browsers such as Google Chrome, Mozilla, Safari or Microsoft Edge. In addition to HTML5 there will be also used two use two frameworks that will be: Bootstrap + Vue and jQuery.js

In addition to the web application, an application will also be developed for the Android operating system because a large number of the devices currently in use are mobile devices and the vast majority use Android. This application will be created using the Android Apache Cordova application development environment.

Therefore, this solution will allow to execute, in different devices, a generic simulator to program in assembler without the need to make installations or internet connection.

B.2. **Simulator Architecture**

The simulator to be developed is made up of different components that will perform a specific function in the system. These are:

- Architecture editor: allows defining and modifying the architecture to be used in the simulation.
- Compiler: is responsible for compiling the assembly code entered so that it can be executed.
- Simulation kernel: this component is in charge of executing the compiled code on the defined architecture.
- User interface: this component is in charge of showing on screen the current status of the simulator, as well as capturing the actions that the user performs on it.

Below is a diagram in which you can see the different components that make up the simulator and the interactions between them:

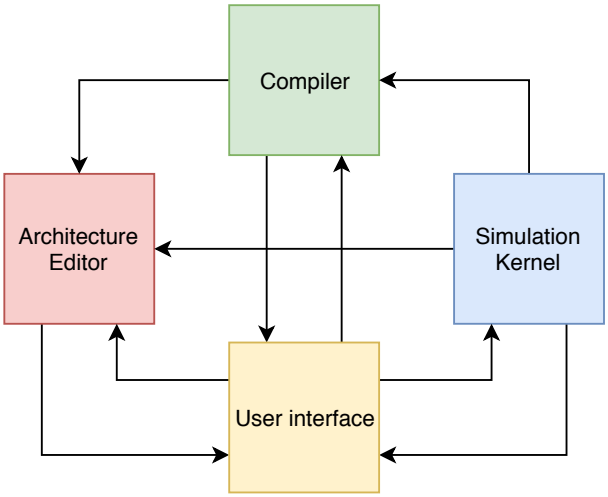


Fig. B.1. Application Architecture

The architecture editor allows you to define and modify the different characteristics of the architecture in which you are going to simulate the assembly code entered by the user. Some of the features are:

- Sets of registers: contain a group of registers that have similar characteristics, such as record type.
- Registers: they are in charge of storing a numeric value inside, which can be an integer or decimal value.
- Instructions: perform arithmetic operations, system calls or memory accesses during the simulation of an assembly program.
- Pseudoinstructions: These are complex instructions that are divided into several instructions in order to carry out the desired action.
- Directives: are commands used to indicate to the compiler a segment, symbols, subroutines, etc.

These characteristics are essential to make a simulation that is similar to reality, omitting others that would make this definition and simulation too complex for students.

The compiler is an essential component for the operation of the simulator, as it validates the assembly code entered by the user and stores in memory the instructions to

be executed, as well as the data. This component will also allow to generate a file with the code compiled in binary.

If the code entered is correct, the compiler will load into memory all the instructions and data entered so that this code can be simulated later, but, if an error is found, the compiler will display a warning to inform the user of the error that has occurred and the code line where it is located so that he can correct it.

The simulation kernel is in charge of executing the assembly code that the user has introduced in the application and that the compiler has validated and stored in memory.

To perform the execution, this component must take into account the definition of the architecture, since the definition of each instruction indicates the operations performed by the instruction and is the attribute that the simulation kernel uses to perform its function. In addition, you must also use the architecture description to verify that the instruction can read and write in the registers that the instruction uses in each of the actions and, if a register cannot be read or written, this component will notify the user and stop the execution of the program. Additionally, this component will also be able to detect if the instruction definition is correct or contains syntax errors and, in this case, it will also notify the user and stop execution.

This component may simulate the instruction-by-instruction program or the entire program, stopping the execution when it reaches the end or when it finds a breaking point.

The user interface will be the component in charge of showing the current status of the simulator to the user, as well as being the component on which the user interacts to perform actions on the components explained above. The user interface will be formed by three different screens that will be directly related to one of the previous components.

C. Evaluation

In order to evaluate the developed simulator, the error messages that it shows to the user when the assembly code that is compiled contains some error, will be compared and, in addition, the feedback offered to users during the execution of a program will also be compared, in order to verify that CREATOR is a more didactic simulator than the existing

ones. For this evaluation, MARS will be used since it is one of the most used simulators to recreate the operation of an assembly program. In order to do this, five different tests will be carried out where it will be possible to observe how they notify the errors produced by each tool and how the feedback is.

The first test carried out was to introduce in a word type data a word instead of a numeric value. The results obtained in both simulators were as follows:

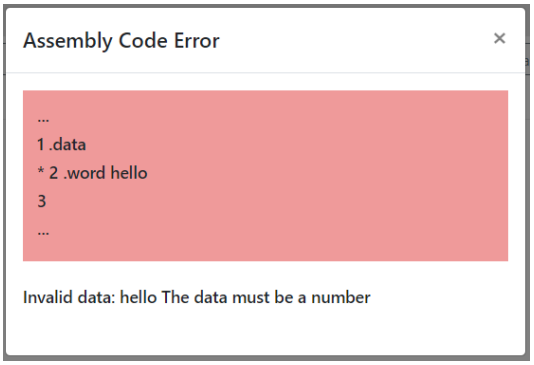


Fig. C.1. Evaluation 1 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 8: Symbol "hello" not found in symbol table.
Assemble: operation completed with errors.
```

Fig. C.2. Evaluation 1 MARS

As can be seen, CREATOR indicates the error more accurately than MARS, allowing users to better understand the error that has occurred and thus correct it more quickly.

The next test is to insert an instruction that is not defined in the tool. The results are shown below:

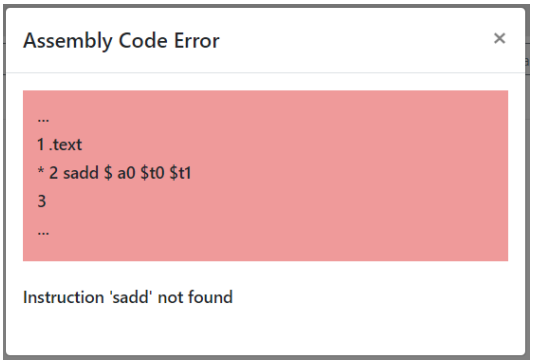


Fig. C.3. Evaluation 2 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 2: "sadd" is not a recognized operator
Assemble: operation completed with errors.
```

Fig. C.4. Evaluation 2 MARS

As in the previous comparison, the CREATOR result is more intuitive than the one shown by MARS.

The third test to be performed consists of entering the name of a record that is not defined in the simulator. The results obtained are:

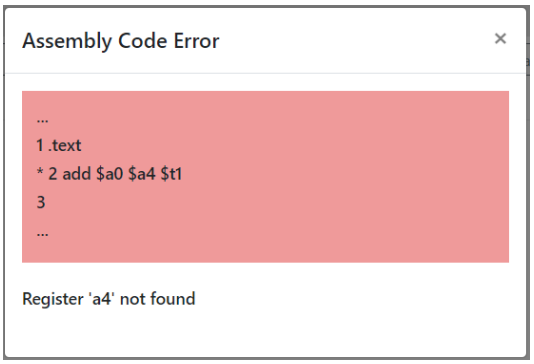


Fig. C.5. Evaluation 3 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Error in D:\Users\Diego\Downloads\mips1.asm line 2 column 10: "$a4": operand is of incorrect type
Assemble: operation completed with errors.
```

Fig. C.6. Evaluation 3 MARS

This test follows the same dynamics as in the two previous cases.

The fourth test is to introduce a directive that is not defined in the tool. The result obtained in both tools is:

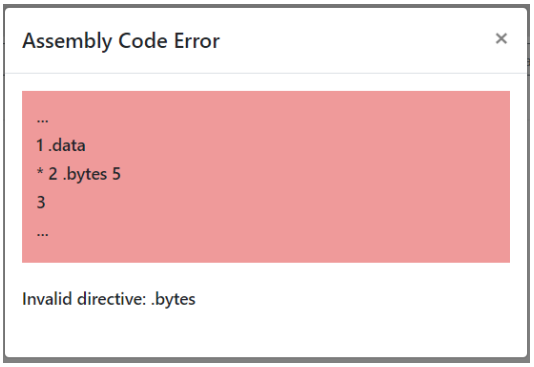


Fig. C.7. Evaluation 4 CREATOR

```
Assemble: assembling D:\Users\Diego\Downloads\mips1.asm

Warning in D:\Users\Diego\Downloads\mips1.asm line 2 column 2: MARS does not recognize the .bytes directive.  Ignored.
Assemble: operation completed successfully.
```

Fig. C.8. Evaluation 4 MARS

As you can see, CREATOR notifies the user of the error in detail. MARS, however, compiles the entire program ignoring the directive entered. This is a problem for users as they are entering an erroneous assembly code, but the simulator does not correctly report the error and this error can be ignored.

The last comparison made between these two simulators consists of analyzing how each simulator indicates the instruction that has been executed and the instruction that will be executed in the next step. The results obtained are:

Break	Address	Label	User Instructions	Loaded Instructions
	0x0	main	beq $5 $a1 b	beq $5 $a1 0x10
	0x4	c	addi $a0 $a1 10	addi $a0 $a1 10
	0x8		addi $a0 $a1 100	addi $a0 $a1 100
	0xc		mul $4 $a0 $a0	mul $4 $a0 $a0
	0x10	b	addi $a0 $a1 15265	addi $a0 $a1 15265
	0x14		addi $a0 $a1 105525452	lui $at 0x064A
	0x18			ori $at $at 0x30CC
	0x1c			add $a0 $a1 $at
	0x20		move $4 $0	add $4 $r0 $0

Fig. C.9. Evaluation 5 CREATOR

Bkpt	Address	Code	Basic		Source
	0x00400000	0x10a50003	beq $5,$5,0x00000003	5:	beq $5 $a1 b
	0x00400004	0x20a4000a	addi $4,$5,0x0000000a	6:	c: addi $a0 $a1 10 #oihewfolfjoijioerg
	0x00400008	0x20a40064	addi $4,$5,0x00000064	7:	addi $a0 $a1 100
	0x0040000c	0x70842002	mul $4,$4,$4	8:	mul $4 $a0 $a0
	0x00400010	0x20a43ba1	addi $4,$5,0x00003ba1	9:	b: addi $a0 $a1 15265
	0x00400014	0x3c01064a	lui $1,0x0000064a	10:	addi $a0 $a1 105525452
	0x00400018	0x342130cc	ori $1,$1,0x000030cc		
	0x0040001c	0x00a12020	add $4,$5,$1		
	0x00400020	0x00002021	addu $4,$0,$0	11:	move $4 $0

Fig. C.10. Evaluation 5 MARS

As can be seen in the previous illustrations, CREATOR indicates, at execution time, the instruction that has just been executed (blue) and the next instruction to be executed (green), while MARS only shows the next instruction to be executed.

After carrying out these five tests, it has been possible to verify that the developed tool is more intuitive and didactic than others that already exist and that are very used, since the errors that the user makes when writing an assembly program are indicated in a more precise way, which is of great help when learning. In addition, CREATOR, when executing a program, shows both the last instruction executed and the next one to be executed. This is an important aspect, especially in the case that the program contains jumps, since it allows to see them quickly and thus to understand in a more efficient way the operation of the program and even to debug the program in front of possible errors of implementation that could have been produced.

D. Conclusions and future work

D.1. Conclusions

At this point, we are going to consider whether or not the objectives proposed in the Section A.2 have been met.

The first objective that was proposed was that the simulator would allow to define different sets of instructions, to define the architecture and that the simulator would be generic enough to be able to define any set of instructions. In order to achieve this objective, a study had to be made of the attributes that form the instructions, pseudoinstructions and registers of different architectures in order to select those that are common and essential for a correct definition of the component. In addition, it was also necessary to study the organization of the data structure that stores all the components that make up an architecture so that it would be compact, efficient in searches and easily editable, due to the fact that this is a very important characteristic of the tool developed. Therefore, as can be seen in this report this objective has been achieved.

The second objective was that the simulator should allow programs to be written in assembly language using the available instruction set. To achieve this goal, an assembly language compiler had to be designed and implemented. This compiler is responsible for verifying that the written code corresponds to the set of instructions that has been loaded into the simulator and, if this is not the case, notify the user of the error produced. As you

can see in the previous chapters, this objective has been achieved.

The third objective indicates that the tool must be able to compile programs written in assembly language. A compiler was designed and implemented for this, as for the previous objective. Therefore, the objective has been fulfilled as can be seen throughout this report.

The following objective presented indicates that the tool would allow to execute and debug programs in assembler. In order to meet this objective, a module was developed that is in charge of this functionality. To carry out the execution, the instructions and data must be obtained from the simulator memory and, by defining the instruction, the corresponding operations are carried out. In order to allow the debugging of the execution of the program, several views were created in the user interface that allow to see the execution statistics, the values stored in the main memory and in the registers, besides, in this last case, the stored values are shown in different formats. Finally, this objective has also been achieved.

The fifth objective states that it should be possible to visualize the state of the computer after executing each one of the instructions of the program. In both cases, the value stored in the registers and main memory can be displayed at execution time, since these values are updated after each instruction has been executed. As can be seen in this memory, this objective is met.

The last objective presented was that the simulator should allow to visualize and edit the different attributes of the components that make up the architecture. To do this, several views were created in the user interface that showed all the components and their attributes and, in order to edit the attributes, different forms were created that allowed modifying their values. This objective has also been achieved as it has been observed throughout this memory.

D.2. **Future work**

Once this project is finished, a series of future works will be presented that will serve to provide this tool with more functionalities. These works are:

- Expand the simulator with other assembler languages such as ARM or assembler

for Intel architecture.

- Incorporate a cache simulator.
- Expand the simulator with functions to be used by teachers to correct the practices.

Glosario

biblioteca Es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.. 18, 31, 45, 46, 57, 58, 111, 130, 147, 162

binario Es un archivo informático que contiene información de cualquier tipo codificada en binario para el propósito de almacenamiento y procesamiento en ordenadores.. 6, 15, 31, 57, 58, 88, 130

ensamblador Es un lenguaje de programación de bajo nivel.. 2, 3, 5–10, 12, 13, 15, 23, 24, 29–31, 33, 35, 37, 43–49, 55–59, 62, 69, 74, 75, 81–83, 85, 87, 88, 101, 103, 108–113, 115–120, 123, 127–132, 134–140, 142, 143, 147, 153, 156–159, 161–164

framework Es un conjunto estandarizado de conceptos, prácticas y criterios para una problemática particular.. 17–19, 21, 69, 72–74, 85, 86

hardware Conjuntos de componentes físicos que componen una computadora.. 9, 157

pipeline Es una técnica que consiste en transformar un flujo de datos en un proceso formado por carias fases secuenciales.. 9, 10

software Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.. 4, 23, 24, 69, 70, 87, 99, 146, 157

Siglas

Ajax Asynchronous Javascript And XML. 18

ALU Arithmetic Logic Unit. 78

API Application Programming Interface. 19–21

ARCOS Grupo de Arquitectura de Computadores. 87

ARM Advanced RISC Machine. 10–13, 23, 28, 50, 86, 158

CREATOR didaCtic geneRic assEmbly progrAmming simulaTOR. 2, 140–143, 159

CSS Cascading Style Sheets. 15, 16, 18–21, 66, 72, 74, 85

DOM Document Object Model. 18, 72, 73

ECMA European Computer Manufacturers Association. 17

GCC GNU Compiler Collection. 13

GNU GNU Not Unix. 13, 69, 70, 153

HDD Hard Disk Drive. 151

HTML HyperText Markup Language. 15–21, 66, 72, 74, 85, 86

IDE Integrated Development Environment. 8

IEEE Institute of Electrical and Electronics Engineers. 24

IVA Impuesto sobre el Valor Añadido. 152

JSON JavaScript Object Notation. 20, 62, 73

LGPL Licencia Pública General Menor de GNU. 70, 153

LOPDGDD Ley Organica de Protección de Datos Personales y Garantia de los Derechos Digitales. 69

MARS MIPS Assembler and Runtime Simulator. 5, 7, 8, 140–143

MIPS Microprocessor without Interlocked Pipeline Stages. 5–7, 9, 23, 28, 50, 86, 157

MIT Massachusetts Institute of Technology. 69

MVC Modelo Vista Controlador. 18, 72

MVVM Modelo-Vista Vista-Modelo. 72, 73

RAM Random Access Memory. 99, 151

RGPD Reglamento General de Protección de Datos. 69

RISC Reduced Instruction Set Computing. 7

SSD Solid-State Drive. 151

W3C World Wide Web Consortium. 15

WYSIWYG What You See Is What You Get. 7, 8

XML Extensible Markup Language. 73

Bibliografía

[1] J. Larus. (). SPIM: A MIPS32 Simulator, [En línea]. Disponible en: `http://spimsimulator.sourceforge.net/` (Acceso: 31-03-2019).

[2] K. Vollmar y P. Sanderson, "MARS: an education-oriented MIPS assembly language simulator", en *SIGCSE*, vol. 6, 2006, pp. 239-243.

[3] I. Branovic, R. Giorgi y E. Martinelli, "WebMIPS: a new web-based MIPS simulation environment for computer architecture education", en *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, ACM, 2004, p. 19.

[4] (). ASP.NET, [En línea]. Disponible en: `https://dotnet.microsoft.com/apps/aspnetl` (Acceso: 03-04-2019).

[5] (). VisUAL A highly visual ARM emulator, [En línea]. Disponible en: `https://salmanarif.bitbucket.io/visual/index.html` (Acceso: 01-04-2019).

[6] S. B. Mir, G. F. Llueca, J. C. F. Fernández y G. L. Navarro, "ARMSim y QtARMSim: simulador de ARM para docencia", en *JENUI 2015*, Universitat Oberta La Salle, 2015, pp. 2-9.

[7] M. Wooldridge, *HTML5*. Indianapolis, Ind.: Indianapolis, Ind. : John Wiley & Sons, Inc, 2011, ID: 34UC3M_ALMA51193279930004213.

[8] T. Negrino, *JavaScript*. Place of publication not identified Peachpit Press, 2012, ID: 34UC3M_ALMA51223934400004213.

[9] (). Acerca de JavaScript, [En línea]. Disponible en: `https://developer.mozilla.org/es/docs/Web/JavaScript/Acerca\_de\begin{\_}content\end{\_}JavaScript` (Acceso: 18-03-2019).

[10] J. J. Gutiérrez, "¿Qué es un framework web?", *Available in: http://www.lsi.us.es/javierj/investigacion_ficheros/Framework.pdf Accessed May*, vol. 12, 2014.

[11] (). Twitter, [En línea]. Disponible en: https://twitter.com (Acceso: 03-04-2019).

[12] S. Moreto, *Bootstrap by example : master Bootstrap 4's frontend framework and build your websites faster than ever before*. Birmingham : Packt Publishing, 2016, ID: 34UC3M_ALMA51214995280004213.

[13] S. Holzner, *jQuery*. Berkeley, Calif.: Berkeley, Calif. : Peachpit Press, 2009, ID: 34UC3M_ALMA51231789390004213.

[14] C. E. jQuery, *jQuery cookbook*, First edition. Beijing; Sebastopol, California : O'Reilly Media, 2009, ID: 34UC3M_ALMA51193396360004213.

[15] *React*. [En línea]. Disponible en: https://reactjs.org/.

[16] A. Q. Haviv, *MEAN web development : master real-time web application development using a mean combination of MongoDB, Express, Angular JS, and Node.js*. Birmingham, England : Packt Publishing, 2014, ID: 34UC3M_ALMA51207528200004213.

[17] (). Vue.js, [En línea]. Disponible en: https://vuejs.org/ (Acceso: 28-03-2019).

[18] A. Freeman, *Pro Vue.js 2*. Berkeley, CA : Apress : Imprint: Apress, 2018, ID: 34UC3M_ALMA51239231840004213.

[19] (). Bootstrap + Vue, [En línea]. Disponible en: https://bootstrap-vue.js.org/ (Acceso: 28-03-2019).

[20] H. Saleh, *JavaScript mobile application development : create neat cross-platform mobile apps using apache Cordova and jQuery mobile*. Birmingham, England : Packt Publishing, 2014, ID: 34UC3M_ALMA51207998000004213.

[21] R. Khanna, *Getting started with Ionic : get up and running with developing effective hybrid mobile apps with Ionic*. Birmingham, England; Mumbai, India : Packt Publishing, 2016, ID: 34UC3M_ALMA51213903950004213.

[22] *IEEE Guide for Software Requirements Specifications*, ID: 1, 1984. DOI: 10.1109/IEEESTD.1984.119205.

[23] (). MIT License, [En línea]. Disponible en: https://opensource.org/licenses/MIT (Acceso: 10-04-2019).

[24] (). Licencias - Proyecto GNU, [En línea]. Disponible en: https://www.gnu.org/licenses/licenses.html (Acceso: 10-04-2019).

[25] (). Apache Licenses, [En línea]. Disponible en: http://www.apache.org/licenses/ (Acceso: 10-04-2019).

[26] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*.

[27] BOE, *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.*

[28] (). XML, [En línea]. Disponible en: https://www.w3.org/XML/ (Acceso: 09-05-2019).

[29] (). JSON, [En línea]. Disponible en: http://json.org/ (Acceso: 09-05-2019).

[30] (). Android Studio, [En línea]. Disponible en: https://developer.android.com/studio (Acceso: 09-05-2019).

[31] (). Apache Cordova, [En línea]. Disponible en: https://cordova.apache.org/ (Acceso: 09-05-2019).

[32] F. G. Carballeira, J. C. Pérez, J. D. G. Sánchez y D. E. Singh, *Problemas resueltos de estructura de computadores*, [2ª ed.]. Madrid: Madrid : Paraninfo, 2015, ID: 34UC3M_ALMA21164941540004213.

[33] H. D. Benington, "Production of Large Computer Programs", *Annals of the History of Computing*, vol. 5, n.º 4, pp. 350-361, 1983, ID: TN_ieee_s4640770. DOI: 10.1109/MAHC.1983.10102.

[34] T. Grimm, "The human condition: a justification for rapid prototyping", *Time Compression Technologies*, vol. 3, n.º 3, pp. 1-6, 1998.

[35] B. W. Boehm, "A spiral model of software development and enhancement", *Computer*, vol. 21, n.º 5, pp. 61-72, 1988, ID: TN_ieee_s59. DOI: 10.1109/2.59.

[36] BOE, *Ley 37/1992, de 28 de diciembre, del Impuesto sobre el Valor Añadido.* 1992.