

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Grado en Ingeniería Informática



Trabajo Fin de Grado

WepSIM: Simulador de procesador elemental con unidad de control microprogramada

Autor: Javier Prieto Cepeda
Tutor: Félix García Carballeira

Leganés, Madrid, España
Junio 2017

*If you would not be forgotten as soon as you
are dead and rotten, either write things worth
reading, or do things worth the writing.*

Benjamin Franklin

Agradecimientos

Es muy difícil incluir en tan pocas palabras a todas aquellas personas que han puesto su granito de arena a lo largo de estos años para que a día de hoy, haya logrado llegar hasta aquí. Es por ello que si me dejo a alguien sin incluir, me disculpe y no dude en sentirse parte de estas palabras.

En primer lugar, quiero agradecer de corazón a mis padres Tomás y Raquel el haberme educado y enseñado a elegir aquel camino que me hiciera feliz de verdad, dándome la oportunidad de estudiar con su esfuerzo y sacrificio. Aquí también estás tu David, pese a llegar unos años después de mí, no paras de enseñarme día a día nuevas lecciones, de los pequeños también se aprende. Gracias.

También debo de dar las gracias a la persona que ha tenido que aguantar estos 5 años las consecuencias de estudiar esta carrera. Míriam, gracias por tu paciencia, tus consejos y tus ánimos.

Mis abuelos también han sido parte de esas personas que han sumado su granito de arena enseñándome y aconsejándome en todo momento, ejerciendo de guías. Sé que vosotros habéis sufrido también mucho en estos años, pero aquí está el premio.

Por otro lado, debo de agradecer a dos personas el darme la oportunidad de realizar este proyecto. Félix y Álex, gracias por este tiempo en el que tanto he podido aprender y en el que tanto me habéis ayudado.

También son parte de este proyecto los grandes amigos que he hecho en la travesía por el grupo de investigación ARCOS. Uno de ellos además, después de compartir asiento en clase. Saúl, gracias por tu ayuda y por los buenísimos momentos que hemos compartido. Carlos, nos quedan más carreras por hacer juntos. Fran, Estefanía, Silvina, Cascajo, Jesús Cristina, Garci, Pablo, David, Rafa y Alfredo, gracias por acogerme tan bien en el laboratorio y por ayudarme en todo momento.

Agradecer también a Jesús, Javi Blas y al resto de personas de componen el grupo de investigación ARCOS la ayuda prestada durante este tiempo.

Por otro lado, también quiero destacar a aquellos compañeros de prácticas y amigos que he hecho a lo largo de estos años en la carrera, y que siempre recordaré. Sergio, Planet, Álex, Guille, Juanlu, Sandra y Marin.

Por último, me gustaría dar las gracias también a todas aquellas personas que forman el sistema de educación pública en España, ya que sin su esfuerzo y dedicación, no podríamos llegar hasta aquí.

**WepSIM: Simulador de procesador elemental con unidad de control
microprogramada**

by

Javier Prieto Cepeda

Abstract

Keywords: MIPS · Simulation · Assembler · Microprogramming

Supervisor: Félix García Carballeira

Title: Full Professor

Índice general

<i>Agradecimientos</i>	v
Abstract	vii
Contents	xi
Índice de figuras	xiv
Índice de tablas	xviii
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 para la enseñanza de la Estructura y Arquitectura de Computadores	5
2.1.1 Simuladores para microprogramación	6
2.1.2 Simuladores para programación en ensamblador	6
2.1.3 Propuesta de simulación unificada	13
2.2 Tecnologías web	14
2.2.1 HTML5	14
2.2.2 Frameworks	16
2.3 Procesador elemental WepSIM	17
3 Análisis	21
3.1 Descripción del proyecto	21

3.2	Requisitos	22
3.2.1	Requisitos de Usuario	25
3.2.2	Modelo de Casos de Uso	31
3.2.3	Requisitos Funcionales	40
3.2.4	Requisitos No-Funcionales	49
3.3	Marco Regulador	52
3.3.1	Aspectos Legales	52
4	Diseño	53
4.1	Estudio de la solución final	53
4.1.1	Solución elegida	55
4.2	Arquitectura de WepSIM	55
4.2.1	Modelo hardware	57
4.2.2	Modelo software	59
4.2.3	Kernel del simulador	62
5	Implementación y despliegue	63
5.1	Implementación	63
5.2	Despliegue	70
6	Verificación, validación y evaluación	75
6.1	Verificación y validación	75
6.1.1	Pruebas de verificación	76
6.1.2	Pruebas de validación	87
7	Planificación y presupuesto	99
7.1	Planificación	99
7.1.1	Justificación de la Metodología	99
7.1.2	Ciclo de Vida	100
7.1.3	Tiempo Estimado	101
7.2	Presupuesto	102
7.2.1	Coste del Proyecto	102
7.2.2	Oferta de Proyecto Propuesta	105
7.3	Entorno Socio-Económico	106

8 Conclusiones y trabajos futuros	109
8.1 Conclusiones	109
8.1.1 Contribuciones	110
8.1.2 Publicaciones	111
8.2 Trabajos Futuros	112
Glossary	113
Acronyms	113
Bibliography	115

Índice de figuras

2-1	Ejemplo de definición de microcódigo en simulador Tanenbaum.	7
2-2	Interfaz SPIM.	8
2-3	Interfaz QtSPIM.	8
2-4	Interfaz MARS.	10
2-5	Interfaz em88110.	11
2-6	Interfaz WebMIPS.	12
2-7	Arquitectura CPU WepSIM	17
2-8	Arquitectura unidad de control WepSIM.	18
3-1	Diagrama Modelo Casos de Uso 1.	31
3-2	Diagrama Modelo Casos de Uso 2.	32
4-1	Arquitectura de WepSIM.	56
4-2	Modelado del hardware.	57
4-3	Ejemplo de modelado de una puerta triestado.	58
4-4	Ejemplo de modelado de un registro.	58
4-5	Ejemplo de formato de instrucción.	60
4-6	Ejemplo de código fuente en ensamblador.	60
4-7	Formato de instrucción descrito en el microcódigo y ejemplo de su traducción en binario.	61
5-1	Estructura de ficheros.	71
5-2	Dependencias entre ficheros.	71
6-1	Verificación y validación software.	76

7-1 Modelo en Espiral (Boehm, 2000). 101

7-2 Diagrama Gantt. 102

Índice de tablas

2.1	Comparación de simuladores de ensamblador y microcódigo.	13
3.1	Plantilla para la especificación de requisitos.	24
3.2	Requisito de usuario UR-C01.	25
3.3	Requisito de usuario UR-C02.	25
3.4	Requisito de usuario UR-C03.	26
3.5	Requisito de usuario UR-C04.	26
3.6	Requisito de usuario UR-C05.	26
3.7	Requisito de usuario UR-C06.	27
3.8	Requisito de usuario UR-C07.	27
3.9	Requisito de usuario UR-C08.	27
3.10	Requisito de usuario UR-C09.	28
3.11	Requisito de usuario UR-R01.	28
3.12	Requisito de usuario UR-R02.	29
3.13	Requisito de usuario UR-R03.	29
3.14	Requisito de usuario UR-R04.	29
3.15	Requisito de usuario UR-R05.	30
3.16	Requisito de usuario UR-R06.	30
3.17	Requisito de usuario UR-R07.	30
3.18	Plantilla de caso de uso.	33
3.19	Caso de Uso UC-01.	33
3.20	Caso de Uso UC-02	34
3.21	Caso de Uso UC-03.	34
3.22	Caso de Uso UC-04.	35

3.23 Caso de Uso UC-05.	35
3.24 Caso de Uso UC-06.	36
3.25 Caso de Uso UC-07.	36
3.26 Caso de Uso UC-08.	37
3.27 Caso de Uso UC-09.	37
3.28 Caso de Uso UC-10.	38
3.29 Caso de Uso UC-11.	38
3.30 Caso de Uso UC-12.	39
3.31 Caso de Uso UC-13.	39
3.32 Requisito Funcional SR-F-F01.	40
3.33 Requisito Funcional SR-F-F02.	40
3.34 Requisito Funcional SR-F-F03.	41
3.35 Requisito Funcional SR-F-F04.	41
3.36 Requisito Funcional SR-F-F05.	41
3.37 Requisito Funcional SR-F-F06.	42
3.38 Requisito Funcional SR-F-F07.	42
3.39 Requisito Funcional SR-F-F08.	43
3.40 Requisito Funcional SR-F-F09.	43
3.41 Requisito Funcional SR-F-F10.	44
3.42 Requisito Funcional SR-F-F11.	44
3.43 Requisito Funcional SR-F-F12.	44
3.44 Requisito Funcional SR-F-F13.	45
3.45 Requisito Funcional SR-F-F14.	45
3.46 Requisito Funcional SR-F-F15.	45
3.47 Requisito Funcional SR-F-F16.	46
3.48 Requisito Funcional SR-F-F17.	46
3.49 Requisito Funcional SR-F-F18.	47
3.50 Requisito Funcional SR-F-F19.	47
3.51 Requisito Funcional SR-F-F20.	47
3.52 Requisito Funcional SR-F-F21.	48
3.53 Requisito Funcional SR-F-F22.	48
3.54 Requisito Funcional SR-NF-PL01.	49

3.55 Requisito Funcional SR-NF-PL02.	49
3.56 Requisito Funcional SR-NF-PL03.	50
3.57 Requisito Funcional SR-NF-PL04.	50
3.58 Requisito Funcional SR-NF-PL05.	50
3.59 Requisito Funcional SR-NF-UI01.	51
3.60 Requisito Funcional SR-NF-P01.	51
4.1 Comparación de frameworks y bibliotecas.	54
6.1 Plantilla de pruebas de verificación.	77
6.2 Test de verificación VET-01.	78
6.3 Test de verificación VET-02.	78
6.4 Test de verificación VET-03.	79
6.5 Test de verificación VET-04.	79
6.6 Test de verificación VET-05.	80
6.7 Test de verificación VET-06.	80
6.8 Test de verificación VET-07.	81
6.9 Test de verificación VET-08.	82
6.10 Test de verificación VET-09.	82
6.11 Test de verificación VET-10.	83
6.12 Test de verificación VET-11.	83
6.13 Test de verificación VET-12.	84
6.14 Matriz de trazabilidad de pruebas de verificación.	86
6.15 Plantilla para test de validación.	87
6.16 Test de validación VAT-01.	88
6.17 Test de validación VAT-02.	89
6.18 Test de validación VAT-03.	90
6.19 Test de validación VAT-04.	91
6.20 Test de validación VAT-05.	92
6.21 Test de validación VAT-06.	93
6.22 Test de validación VAT-07.	94
6.23 Test de validación VAT-08.	94
6.24 Test de validación VAT-09.	95

6.25	Test de validación VAT-10.	95
6.26	Test de validación VAT-11.	96
6.27	Matriz de trazabilidad de pruebas de validación.	97
7.1	Información del Proyecto.	103
7.2	Costes de recursos humanos.	103
7.3	Costes de equipamiento.	104
7.4	Otros costes directos.	105
7.5	Resumen de costes.	105
7.6	Oferta propuesta.	106

Capítulo 1

Introducción

El primer capítulo introduce brevemente el objetivo del proyecto, incluyendo las características clave del proyecto y su motivación (Section 1.1, *Motivación*), los objetivos del proyecto (Section 1.2, *Objetivos*), y toda la estructura del documento (Section 1.3, *Estructura del documento*).

1.1 Motivación

La enseñanza de la arquitectura de un computador es una parte básica y fundamental en la formación de los estudiantes de Ingeniería Informática mediante la cual los alumnos logran obtener una visión y comprensión del comportamiento a bajo nivel de la máquina. Para lograr que los estudiantes comprendan y asienten correctamente los fundamentos teóricos, es necesario el uso de clases prácticas, en donde el alumno pueda ser capaz de interactuar con un computador de arquitectura igual o similar a la explicada en teoría y logre extrapolar los fundamentos teóricos al comportamiento real de la máquina.

Uno de los principales problemas a la hora de diseñar estas clases prácticas es lograr obtener los medios necesarios para que los alumnos puedan hacer uso de un computador similar al visto en las clases teóricas, debido al coste que supone tener un número suficiente de computadores para el número de alumnos que deben hacer uso de ellos y su mantenimiento, la limitación de movilidad a la hora de realizar las practicas debido a la necesidad física del computador, etc. Para evitar estos problemas, actualmente se hace uso de simuladores y emuladores que proporcionan las funciones necesarias para las clases prácticas, evitando los problemas anteriormente comentados.

Un emulador es un software que se encarga de imitar el comportamiento de una compu-

tadora de forma que programas pensados para una arquitectura concreta, puedan ser ejecutados en otra diferente. Por otro lado, un simulador es un software que trata reproducir el comportamiento de un computador, pero con un menor nivel de realismo, puesto que el emulador se encarga de modelar de forma precisa el dispositivo de manera que los programas ejecutados sobre él funcionen como si estuvieran siendo ejecutados en el dispositivo original.

Hay distintos simuladores que se pueden utilizar para trabajar con los principales aspectos que se tratan en las asignaturas de Estructura y Arquitectura de Computadores: ensamblador, caché, etc. Aunque la idea de usar distintos simuladores cae dentro de la estrategia de "divide y vencerás", hay dos principales problemas con estos simuladores: cuanto más realistas son más compleja se hace la enseñanza (tanto del simulador como de la tarea simulada), y cuantos más simuladores se usan más se pierde la visión de conjunto.

Hay otro problema no menos importante: la mayoría de los simuladores están pensados para PC. Uno de los objetivos que nos planteamos con WepSIM es que pudiera ser utilizado en dispositivos móviles (smartphones o tablets), para ofrecer al estudiante una mayor flexibilidad en su uso.

Además de tener un simulador portable a distintas plataformas, el simulador ha de ser lo más autocontenido posible de manera que integre la ayuda principal para su uso (no como un documento separado que sirva de manual de uso para ser impreso) permitiendo al usuario hacer un uso completo de la aplicación sin la necesidad de salir de ella.

Por todo ello, nos hemos planteado cómo ofrecer un simulador que sea simple y modular, y que permita integrar la enseñanza de la microprogramación con la programación en ensamblador. En concreto, puede utilizarse para microprogramar un juego de instrucciones y ver el funcionamiento básico de un procesador, y para crear programas en ensamblador basados en el ensamblador definido por el anterior microcódigo. Esto es de gran ayuda, por ejemplo, para la programación de sistemas dado que es posible ver cómo interactúa el software en ensamblador con el hardware en el tratamiento de interrupciones. La idea es ofrecer un simulador que ofrezca una visión global de lo que pasa en hardware y software, evitando además el tiempo extra que supone el aprendizaje de distintas herramientas.

1.2 Objetivos

El objetivo principal de este proyecto, es desarrollar un simulador, que a diferencia de los existentes, pueda simular de forma completa el comportamiento de un procesador elemental permitiendo comprobar el estado de los componentes en cada ciclo de reloj, de manera que ayude a los alumnos a comprender y asimilar de forma sencilla y visual el funcionamiento de un procesador. Los objetivos secundarios son:

- Simular la ejecución de el juego de instrucciones especificado en un computador denominado WepSIM desde el punto de vista de la micropogramación y la programación en ensamblador.
- Permitir la especificación de diferentes juegos de instrucciones.
- Permitir unificar la microprogramación y la programación en ensamblador.
- Permitir al usuario la visualización en cada ciclo de reloj del estado y comportamiento del computador simulado.

1.3 Estructura del documento

El documento contiene los siguientes capítulos:

- Capítulo 1, *Introducción*, presenta una breve descripción del contenido del documento. También incluye la motivación y los objetivos del proyecto.
- Capítulo 2, *Estado del arte*, incluye una descripción de los diferentes tipos de simuladores (de lenguaje ensamblador y de microcódigo) presentando el trabajo relacionado y una descripción de las tecnologías web actuales.
- Capítulo 3, *Análisis*, describe brevemente el proyecto, establece los requisitos y presenta el marco regulador del proyecto.
- Capítulo 4, *Diseño*, , explica la solución elegida, detalla el diseño del sistema, incluyendo todos sus componentes.
- Capítulo 5, *Implementación y despliegue*, incluye los detalles de implementación de las partes principales del software desarrollado y las características necesarias para la implementación de la aplicación.

- Capítulo 6, *Verificación, validación y evaluación*, detalla una verificación y validación completa del proyecto.
- Capítulo 7, *Planificación y presupuesto*, presenta los conceptos relacionados con la planificación seguida, descompone todos los costes del proyecto y describe el entorno socio-económico.
- Capítulo 8, *Conclusiones y trabajos futuros*, incluye las contribuciones del proyecto, explica las principales conclusiones del proyecto y presenta los trabajos futuros.
- Appendix ??, ??, incluye un manual de usuario completo para la aplicación. Contiene un tutorial que guía al usuario desde la creación de un nuevo juego de instrucciones hasta una ejecución completa paso a paso, y una serie de ejemplos educativos para aprender el funcionamiento de un procesador mediante el uso de simulaciones utilizando el software desarrollado.

Capítulo 2

Estado del arte

Este capítulo presenta el estado del arte, la última y más avanzada etapa de las tecnologías relacionadas con nuestra aplicación. En primer lugar se presentan los diferentes simuladores existentes en el ámbito docente tanto para microprogramación (Section 2.1.1) como para la programación en código ensamblador (Section 2.1.2), comparando una comparación de nuestro trabajo con el contexto actual de los distintos simuladores expuestos previamente (Section 2.1.3). Después, se presentan las diferentes tecnologías web más utilizadas en la actualidad 2.2. Por último se describe el procesador en el cual se basa WepSIM.

2.1 Simuladores para la enseñanza de la Estructura y Arquitectura de Computadores

Son muchos los tipos de simuladores existentes en el ámbito docente para la explicación de las distintas partes que componen un computador. Estos simuladores, pueden ser de diferentes ámbitos como por ejemplo, cachés, pipeline, circuitos integrados, microprogramación, programación en ensamblador, etc.

En este apartado, vamos a centrarnos en los tipos de simuladores que tienen relación directa con la propuesta realizada, como lo son los simuladores para la microprogramación y los simuladores para la programación en ensamblador.

2.1.1 Simuladores para microprogramación

En esta sección, se explican los diferentes simuladores existentes para la microprogramación. Cabe destacar, que actualmente existen pocos simuladores que nos proporcionen esta funcionalidad, debido a que una gran parte de ellos, son utilizados de forma interna por las empresas para verificar el correcto funcionamiento de sus unidades en la fase de diseño y desarrollo. En primer lugar, vamos a centrarnos en los simuladores que están más enfocados a una labor docente.

P8080E [1] es un simulador desarrollado en el DATSI en la Facultad de Informática de la Universidad Politécnica de Madrid. Este simulador, no consta de una interfaz gráfica portable e interactiva que se ajuste a las tecnologías actuales. Para realizar poder realizar el desarrollo del microcódigo, el simulador requiere el binario completo de cada ciclo, de forma que resulta un tanto complejo su uso. Pese a ser un simulador bastante completo, no está orientado para la enseñanza de ensamblador y microprogramación con la misma herramienta, puesto que no permite la definición del juego de instrucciones con un formato diferente al definido por defecto.

En [2], se describe el desarrollo y la implementación de un simulador para una arquitectura de microprogramación específica. Fue publicado en el año 1986, y estaba basado en la arquitectura definida en el entonces popular libro de ingeniería de computadores [3]. El simulador, requería de la definición de cada uno de los ciclos de ejecución de las intrucciones para la posterior generación de la memoria de control. Esta definición, se realizaba mediante la sintaxis definida en el libro, lo que hacía de este simulador una herramienta bastante completa al para aprender junto con el libro.

2.1.2 Simuladores para programación en ensamblador

En esta sección, se explican los diferentes simuladores existentes para la programación en ensamblador. Los simuladores más conocidos para labores docentes, son SPIM, MARS y WebMIPS.

SPIM [4], es un simulador de un procesador MIPS de 32 bits desarrollado a principios de 1990. En un primer momento implementaba el juego de instrucciones MIPS-1, utilizado por los computadores MIPS R2000/R3000. Actualmente, implementa la arquitectura MIPS32 más reciente y sus instrucciones adicionales. Permite ejecutar programas en ensamblador para es-

```

                                SOURCE LINE
0:  mar := pc; rd;
1:  pc  := pc + 1; rd;
2:  ir   := mbr;
3:  tir  := lshift(ir + ir);
4:  tir  := lshift(ir);
5:  alu  := tir; if n then goto 9;

6:  mar := ir ; rd;
7:  rd;
8:  ac   := mbr; goto 0;

9:  mar := ir ; mbr := ac; wr;
10: wr; goto 0;
```

Figura 2-1: Ejemplo de definición de microcódigo en simulador Tanenbaum.

ta arquitectura. Además, también permite depurar el código implementado, de forma que el alumno pueda corregir con mayor facilidad los errores cometidos. Este simulador, fue creado por James R. Larus y tiene versiones compiladas para Windows, Mac OS X y Unix/Linux e incluso tiene una versión básica para Android, aunque su diseño no está pensado para dispositivos móviles. Puesto que el simulador está totalmente enfocado al procesador MIPS, posee el juego completo de instrucciones para la versión de 32 bits del procesador y todas las directivas de las que consta el lenguaje. Otra característica que hace de SPIM un potente simulador, es proveer de un pequeño sistema operativo que soporta las principales llamadas al sistema mediante la instrucción syscall. SPIM, es un simulador que permite múltiples configuraciones mediante su interfaz de usuario, de manera que se puede indicar:

- Activación del uso de pseudoinstrucciones.
- Simulación de predicción de saltos y accesos a memoria, con la latencia correspondiente.
- Activación del uso del manejador de la señal trap y la carga del manejador personalizado.
- Activación de la visualización de los mensajes en caso de ocurrir excepciones.
- Activación del uso de "memory-mapped IO".

SPIM, además cuenta con una versión más actualizada del simulador, que posee una interfaz gráfica más potente. Éste simulador se llama QtSPIM [5], y cuenta con todas las características anteriormente mencionadas de SPIM.



QSPin File Simulator Registers Text Segment Data Segment Window Help
✓ User Text QSPin

✓ Kernel Text
✓ Comments
✓ Instruction Value
Data Text

FP Regs Int Regs [16]
Int Regs [16]
Text

```

PC = 0
SPC = 0
SPR = 0
BadAddr = 3000f10
Status = 3000f10

R0 = 0
R1 = 0
R2 = 0
R3 = 0
R4 = 0
R5 = 0
R6 = 0
R7 = 0
R8 = 0
R9 = 0
R10 = 0
R11 = 0
R12 = 0
R13 = 0
R14 = 0
R15 = 0
R16 = 0
R17 = 0
R18 = 0
R19 = 0
R20 = 0
R21 = 0
R22 = 0
R23 = 0
R24 = 0
R25 = 0
R26 = 0
R27 = 0
R28 = 0
R29 = 0
R30 = 0
R31 = 0

```

```

[00040000] 8f640000 lw $4, 0($29)
[00040004] 7a5d0001 addiu $5, $29, 4
[00040008] 24e60004 addiu $6, $5, 4
[0004000c] 00001000 lli $7, $4, 1
[00040010] 00c23021 addiu $6, $6, $2
[00040014] c0c01000 jal 0x00000000 (main)
[00040018] 00000000 nop
[0004001c] 3a020004 ori $2, $0, 10
[00040020] 00000000 syscall

[80000180] 0001a821 addiu $27, $0, $1
[80000184] 3c019000 lui $1, -28672
[80000188] ac221000 lw $2, -1812
[8000018c] 3c019000 lui $1, -28672
[80000190] 3c019000 lui $1, -28672
[80000194] 401a8000 mfc0 $26, $13
[80000198] 00001000 lli $2, $2, 2
[8000019c] 30a00011 andi $4, $4, $1
[800001a0] 00001000 lli $2, $0, 1
[800001a4] 3c019000 lui $4, -28672 [_m1_]
[800001a8] 00000000 syscall
[800001ac] 3a020001 ori $2, $0, 1
[800001b0] 00012004 andi $4, $4, 2
[800001b4] 30a00011 andi $4, $4, $1
[800001b8] 00000000 syscall
[800001bc] 3a020004 ori $2, $0, 4
[800001c0] 334a0034 andi $4, $26, 0
[800001c4] 3c019000 lui $1, -28672
[800001c8] 00001000 ori $1, $1, 24
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop
[800001d4] 00000000 syscall
[800001d8] 3a011001 ori $1, $0, 24
[800001dc] 43a00000 lli $1, $26, 32 [ok, pc-0x0a000001]
[800001e0] 00000000 nop
[800001e4] 40a47000 mfc0 $4, $14
[800001e8] 00001000 ori $4, $4, 3
[800001ec] 00000000 nop
[800001f0] 3a020004 beq $0, $4, 16 [ok, pc-0x800001a0]
[800001f4] 3a020004 ori $2, $0, 10
[800001f8] 00001000 nop
[80000200] 3a020004 ori $2, $0, 4
[80000204] 3c019000 lui $1, -28672
[80000208] 3a020004 ori $4, $1, 13 [_m2_]
[8000020c] 00000000 syscall
[80000210] 00120a82 ori $4, $26, 2
[80000214] 30a00011 andi $4, $4, 3
[80000218] 40a40002 lli $0, $4, 8 [ret-0x80000214]
[8000021c] 401a7000 mfc0 $26, $14
[80000220] 27a00000 addiu $26, $26, 4
[80000224] 40a70000 mfc0 $26, $14
[80000228] 3c019000 lui $1, -28672
[8000022c] 8c220020 ori $2, $12($1)
[80000230] 3c019000 lui $1, -28672
[80000234] 3c024004 lw $4, 516($1)
[80000238] 3c019000 lui $1, $0, 127
[8000023c] 30a00011 andi $4, $4, 3

```

```

User Text Segment [000400000]..[00440000]
1281 lw $a0 0($sp) # argc
1384 addiu $a1 $a0 # argv
1485 addiu $a2 $a1 4 # envp
1586 lli $v0 $a0 2
1687 addiu $a2 $a0 $v0
1788 lli main
1889 nop
1992 lli $v0 10
1993 syscall # syscall 10 (exit)

Kernel Text Segment [80000000]..[80010000]
895 move $t1 $a0 # Save $a0
921 av $v0 $1 # Not re-entrant and we can't trust $ap
931 av $a0 $2 # But we need to use those registers
941 mfc0 $k0 $13 # Cause register
961 andi $a0 $a0 3 # Extract EnoCcode Field
971 andi $a0 $a0 0x1f
981 mfc0 $t1 # syscall 4 (print_str)
1001 la $a0 _m1_
1002 syscall
1051 li $v0 1 # syscall 1 (print_int)
1061 andi $a0 $a0 3 # Extract EnoCcode Field
1071 andi $a0 $a0 0x1f
1081 mfc0 $t1
1101 li $v0 4 # syscall 4 (print_str)
1111 andi $a0 $a0 $a0
1121 lw $a0 _except($a0)

1131 nop
1141 syscall
1161 andi $a0 $a0 0x1f ok, pc # Bad PC exception requires special checks
1171 nop
1191 mfc0 $a0 $16 # EPC
1201 andi $a0 $a0 0x2 # Is EPC word-aligned?
1211 nop
1241 li $v0 10 # Exit on really bad PC
1251 andi $a0 $a0 0x1f
1261 li $v0 4 # syscall 4 (print_str)
1291 la $a0 _m2_
1301 syscall
1321 ori $a0 $a0 2 # Extract EnoCcode Field
1331 andi $a0 $a0 0x1f
1341 hme $a0 $0 ret # 0 means exception was an interrupt
1451 mfc0 $a0 $16 # Dump EPC register
1461 andi $a0 $a0 4 # Skip trailing instruction
1481 mfc0 $a0 $16
1501 li $v0 $1 # Restore other registers
1511 la $a0 $a2
1571 move $a2 $t1 # Restore $a2
1581 mfc0 $a0 $a0 0x1f

```

SPIN version 3.0.3, originally 3.2.1
 Copyright 1990-2010, James R. Larus.
 All Rights Reserved.
 SPIN is distributed under a BSD license.
 See the file README for a full copyright notice.

Figura 2-3: *Interfaz QtSPIM.*

MARS [6], es un simulador con interfaz gráfica desarrollado en JAVA para el lenguaje ensamblador MIPS. Fue creado en el año 2005 como una alternativa del simulador SPIM y diseñado específicamente para las necesidades limitadas de cursos de pregrado. Está basado en la arquitectura MIPS RISC, que tiene un número pequeño de elementos del lenguaje e instrucciones. Consta de operaciones de entrada/salida, saltos, condiciones y operaciones aritmético-lógicas tanto enteras como en coma flotante. Algunas de las mejoras que incorpora MARS con respecto a SPIM son [7]:

- Depuración más sencilla y cómoda, debido a constar de GUI. Esto se debe, a que para añadir un punto de ruptura, únicamente hay que clickar sobre un checkbox que tiene la instrucción.
- Los programas pueden ejecutarse de forma inversa, lo que permite volver a un estado anterior de la máquina en la ejecución de un programa.
- La velocidad de ejecución puede modificarse, de forma que si se ralentiza el usuario puede observar como cada instrucción se resalta cuando está siendo ejecutada y ver los cambios que se producen en los valores del banco de registros y de las ubicaciones de memoria.
- La memoria y los registros se pueden modificar de una manera WYSIWYG.
- Incorpora un editor de texto, eliminando dependencias externas para el uso del simulador.
- Permite que los desarrolladores puedan suministrar nuevos complementos, que por ejemplo, sirvan para extender la arquitectura o mostrar los datos de forma intuitiva.

P88110, es emulador descrito en [8] que se basa en la emulación de un procesador superescalar. El propósito fundamental de este emulador, es mostrar el funcionamiento de este tipo de procesadores, haciendo visible el funcionamiento del pipeline y las dependencias existentes. Está basado en una arquitectura específica (MC88110) y pese a integrar el efecto de un procesador superescalar no integra la microprogramación, que haría de él un emulador completo.

WebMIPS [9], es un simulador desarrollado en la Facultad de Ingeniería de la Información de Sienna, Italia. Está escrito en el lenguaje de programación ASP.NET y se utiliza mediante una página web, de forma que no requiere de su instalación en local. No soporta el juego completo de instrucciones de MIPS, sino que consta de un juego reducido de instrucciones que sirve de introducción al estudio de arquitectura de computador. Con respecto a los anteriores

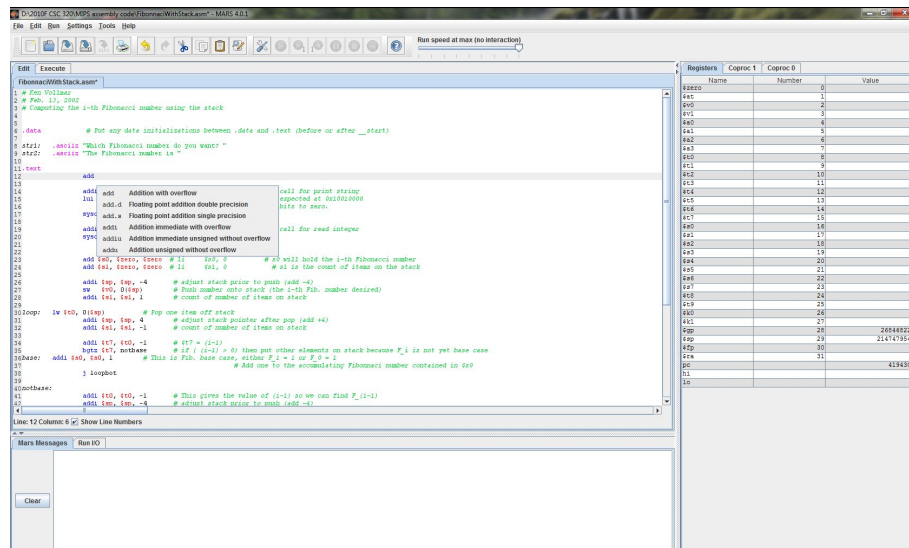


Figura 2-4: Interfaz MARS.

simuladores mencionados, difiere al mostrar el esquema arquitectónico, indicando las 5 etapas de pipeline de las que consta el computador que simula. Otra característica, es la indicación del número de ciclos de reloj que consume la ejecución del programa. Tiene como objetivo principal la demostración de la ejecución del juego de instrucciones básico explicado durante la asignatura. Arquitectura de Computadores impartida por los creadores del simulador.

Entre las diferentes características a destacar de este simulador, se encuentran:

- Verificación del código ensamblador, comprobando que no existen errores en la definición el código.
- Dispone de códigos simples de ejemplo ("load-and-play") que facilitan la comprensión del funcionamiento del procesador.
- Permite diferentes visualizaciones del estado de la memoria y los registros del procesador.
- Consta de dos modos de ejecución: paso a paso y ejecución completa del programa.

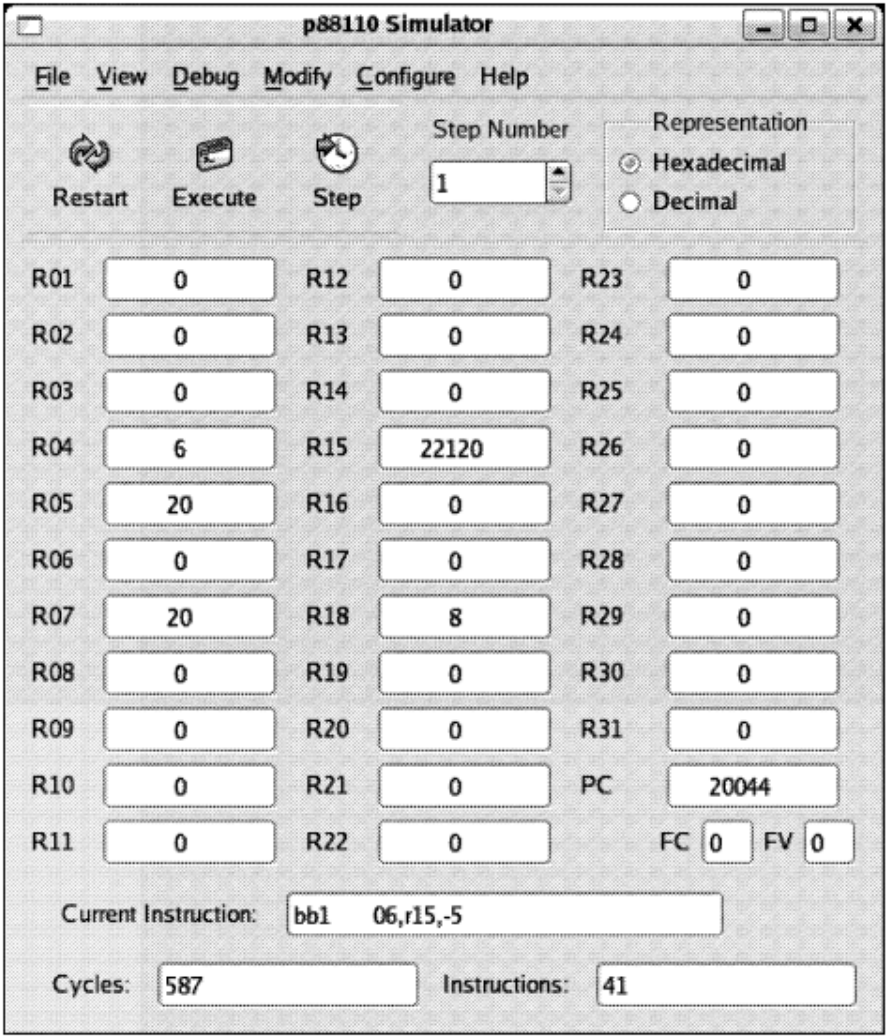


Figura 2-5: Interfaz em88110.

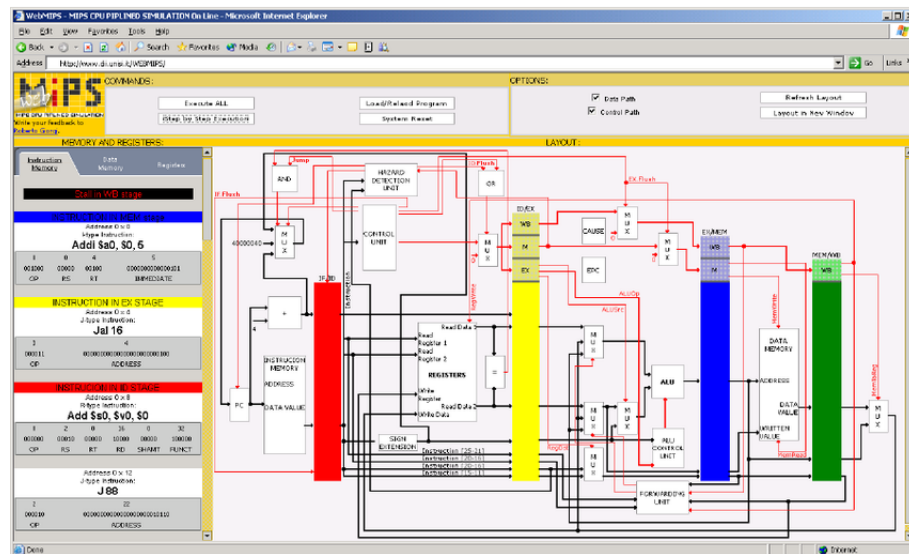


Figura 2-6: Interfaz WebMIPS.

2.1.3 Propuesta de simulación unificada

Hay distintas herramientas muy útiles para labores docentes relacionadas con la asignatura Estructura de Computadores, aunque como se comentó previamente, no existe una herramienta que nos proporcione el desarrollo y simulación tanto a nivel de microcódigo como a nivel de ensamblador. WepSIM, une ambos aspectos ofreciendo:

- Visión interrelacionada de la microprogramación y la programación en ensamblador.
- Flexibilidad en la plataforma usada, pensando en dispositivos móviles.
- Herramienta autocontenida, de forma que incorpora en sí misma tanto la ayuda como distintos ejemplos que favorecen la comprensión del simulador.
- Consta de un modelo hardware que puede ser modificado o ampliado en caso de ser necesario.
- Permite definir un amplio conjunto de instrucciones máquina.
- Puede ser usado como herramienta de microprogramación o de programación en ensamblador.

Simulador	SPIM	MARS	PC88110	WebMIPS	WepSIM	P8080E	MicMac
Ensamblador	✓	✓	✓	✓	✓		
Microprogramación					✓	✓	✓
Multiplataforma	✓			✓	✓		
Interactivo					✓		
Juego de instrucciones personalizable					✓		

Tabla 2.1: Comparación de simuladores de ensamblador y microcódigo.

De este modo, con WepSIM se pretende crear un simulador unificado, abarcando los aspectos más relevantes de la asignatura 'Estructura de Computadores' y permitiendo que todas las prácticas de la asignatura puedan realizarse en la misma plataforma, evitando la pérdida de tiempo y dificultad que supone para el alumno el habituarse a diferentes entornos para cada una de las prácticas.

WepSIM, se puede definir por tanto como un simulador de programación en ensamblador y microprogramación, flexible puesto que permite la personalización del juego de instrucciones

de la máquina, multiplataforma puesto que permite su uso desde diferentes tipos de dispositivos, e interactivo al poder modificar la configuración de la máquina en tiempo de ejecución.

2.2 Tecnologías web

Actualmente, gran parte de los recursos disponibles en el día a día son ofrecidos a través de Internet y su distribución es posible gracias a las tecnologías web. Estas tecnologías, permiten la construcción de las páginas web haciendo uso de las tecnologías para el desarrollo de páginas web y las tecnologías de interconexión de computadores permitiendo a los usuarios el intercambio en formato de hipertexto de todo tipo de datos y de aplicaciones software.

2.2.1 HTML5

Cuando se realiza el desarrollo de una página web, son tres las tecnologías que se emplean con mayor frecuencia: HTML, CSS y JavaScript. Todas estas tecnologías vienen definidas y estandarizadas por el organismo internacional W3C y cada una de ellas, tiene una función concreta en el funcionamiento de la página web:

- **HTML:** Es un lenguaje de marcado con el que se realiza la estructuración de la página web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, entre otros.

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

En cambio, a lo largo de las diferentes versiones, se han añadido y eliminado diferentes características, con la finalidad de hacer el lenguaje más eficiente y facilitar el desarrollo

de páginas web con diferentes plataformas y navegadores. Un navegador web desactualizado, no será capaz de interpretar correctamente una página web escrita en una versión superior de HTML a la que pueda interpretar.

- **CSS:** Es un lenguaje de diseño gráfico utilizado para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado, como puede ser, HTML. Su principal finalidad, es establecer el diseño visual de las páginas web e interfaces de usuario escritas en los lenguajes HTML o XHTML. Este lenguaje además, permite aplicar estilos no visuales como pueden ser las hojas de estilo auditivas.

Esta tecnología web, es usada por la mayoría de sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y GUIs para muchas de las aplicaciones móviles existentes en el mercado. Con ella, se pretende separar el contenido de la página web de la presentación, buscando mejorar la accesibilidad del documento y dando una mayor flexibilidad y control en la especificación de las características de la presentación, permitiendo que varios documentos HTML compartan un mismo estilo usando una única hoja de estilos reduciendo la complejidad y la repetición de código en la estructura del documento.

- **JavaScript:** Es un lenguaje de programación interpretado orientado a las páginas web que surgió de la necesidad de ampliar las posibilidades del HTML. Su principal función en las páginas web, es realizar tareas y operaciones en el marco de la aplicación cliente como la mejora de la interfaz de usuario y la generación de páginas web dinámicas, aunque también es utilizado en el lado del servidor .

Este lenguaje, fue diseñado con una sintaxis similar a la del lenguaje de programación C, aunque adopta nombres y convenciones del lenguaje de programación Java. En cambio, Java y Javascript tienen propósitos y semánticas distintos.

En la actualidad, todos los navegadores realizan la interpretación del código JavaScript integrado en las páginas web. Para poder interactuar con una página web el lenguaje está provisto de una implementación del DOM que facilita el control y manejo de las interacciones del usuario.

2.2.2 Frameworks

Como se indica en [10] las aplicaciones web se han convertido en los últimos años en complejos sistemas software con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, dando servicio a procesos de negocio de considerable envergadura y estableciéndose sobre ellas requisitos estrictos de accesibilidad y respuesta. Para ayudar al desarrollo de estas aplicaciones web, se hace uso de diferentes frameworks y bibliotecas que añaden funcionalidad al entorno de trabajo. Los frameworks son una estructura conceptual para el desarrollo e implementación de aplicaciones web, teniendo como objetivos principales la aceleración del proceso de desarrollo, reutilización de código ya existente y promover buenas prácticas de desarrollo, como el uso de patrones. Entre los frameworks más utilizados en la actualidad, se encuentran:

- **Angular.js:** Es un framework MVC (Modelo-Vista-Controlador) de JavaScript de código abierto mantenido por Google para el desarrollo Web Front End, que permite crear aplicaciones *Single-Page*. Este framework ha supuesto un avance puesto que al implementar el patrón MVC ayuda al desarrollador a separar conceptos. De este modo, adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de un *data binding* bidireccional que permite la sincronización automática de modelos y vistas. Además facilita la creación de test unitarios, lo cual resulta de gran ayuda para los desarrolladores.
- **jQuery.js:** Es una biblioteca de JavaScript rápida, pequeña y característica. Ofrece una infraestructura que proporciona una mayor facilidad para la creación de aplicaciones complejas del lado del cliente. De esta forma, al utilizar esta biblioteca, el desarrollador se asegura que funcionará correctamente en los navegadores de los usuarios.
- **BootStrap:** Es un framework de código abierto para el diseño de aplicaciones web, en el cual se incluyen diferentes plantillas de diseño con tipografía, formularios, botones, menús de navegación y otros elementos de diseño basados en HTML y CSS que proporcionan el desarrollador ayuda a la hora de implementar la interfaz de usuario de su aplicación. Además, es compatible con la mayoría de los navegadores web.
- **Dojo:** Es un framework que contiene APIs que facilitan el desarrollo de aplicaciones web que utilizan la tecnología AJAX, conteniendo un sistema de empaquetado inteligente, efecto de interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente e interacción de APIs con AJAX.

2.3 Procesador elemental WepSIM

WepSIM, es un procesador elemental con unidad de control microprogramada diseñado por el personal del grupo de investigación ARCOS de la Universidad Carlos III de Madrid para la docencia en la asignatura Estructura de Computadores.

En la figura 2-7, se muestra la estructura del Procesador Elemental WepSIM. WepSIM, consta de un módulo de memoria, un dispositivo teclado, un dispositivo pantalla y un dispositivo de E/S genérico que se puede utilizar para trabajar con interrupciones.

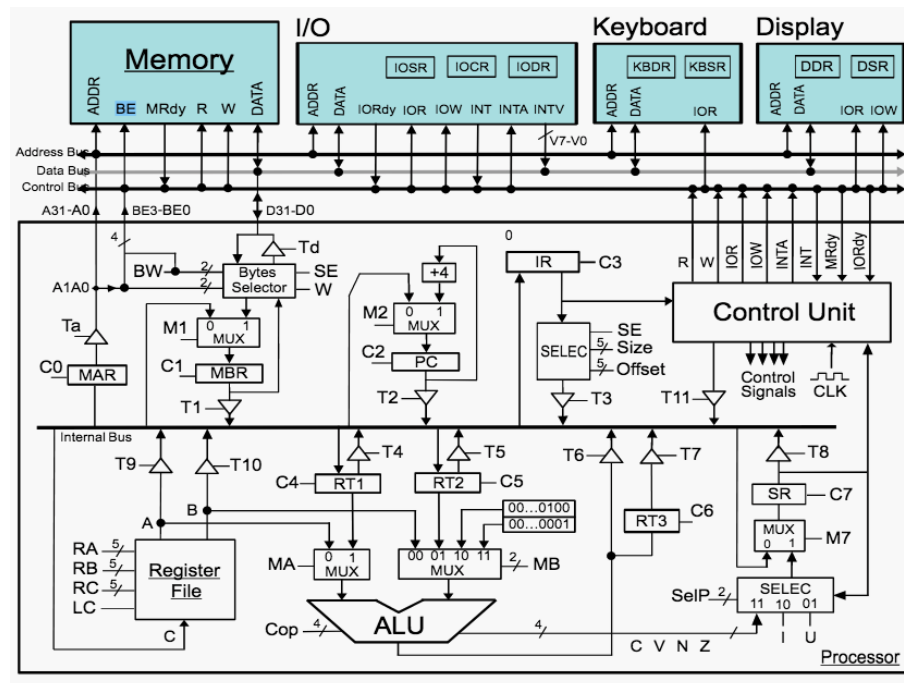


Figura 2-7: Arquitectura CPU WepSIM.

WepSIM es un procesador de 32 bits que direcciona la memoria por bytes, que cuenta con un banco de 32 registros y dos registros adicionales (RT1 y RT2) que no son visibles para el programador de ensamblador, pero que permiten el almacenamiento temporal de datos para la realización de operaciones intermedias. Desde los registros, es posible enviar los valores para operar en una ALU que dispone de las 15 operaciones aritmético-lógicas más comunes. El registro PC tiene su propio operador de sumar cuatro, de forma que no es necesario hacer uso de la ALU para esta operación. El resultado de las operaciones realizadas en la ALU puede ser almacenado en un registro temporal (RT3) que también es invisible para el programador de ensamblador, o ser enviado directamente al bus interno a través del correspondiente triestado.

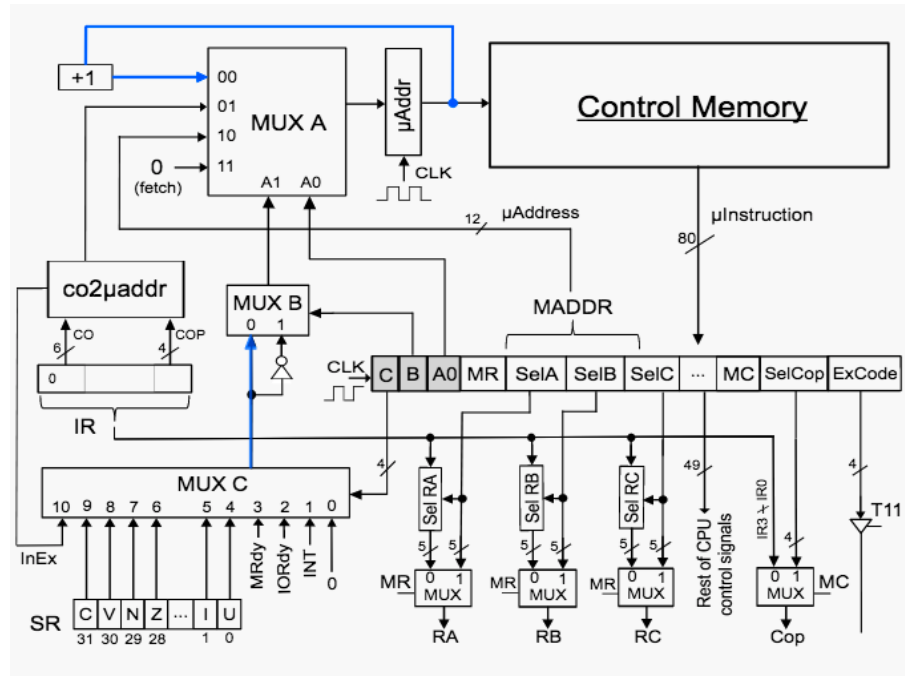


Figura 2-8: Arquitectura unidad de control WepSIM.

El registro de estado (SR) puede ser actualizado con los flags resultantes de la última operación de la alu (O, N y Z). Para ello, SELEC/SelP representa un bloque de circuitos que permite indicar qué parte del registro de estado (SR) debe ser actualizado. A la derecha de SELEC/SelP llegan los bits del registro de estado SR como entrada (Input=O N Z I U) y SelP permite seleccionar qué grupo de estos bits se actualizará en el registro de estado: los bits O, N y Z con los valores procedentes de la ALU, el bit I con el valor indicado o el bit U con el valor indicado para el mismo.

El registro de instrucción (IR) tiene asociado un módulo selector (circuito de más alto nivel que un multiplexor, etc.) que permite seleccionar un segmento del valor binario almacenado en el registro de instrucción que pasará hacia T3.

En concreto, se indica la posición (Offset, donde 0 represente el bit menos significativo del registro IR) inicial y el número de bits (Size) a tomar a partir de dicha posición inicial, así como si se desea hacer extensión de signo (SE) antes de pasar el valor a la entrada de T3.

Los registros MAR y MBR se usan para almacenar la dirección y el contenido asociado a esta dirección en las operaciones de lectura/escritura con la memoria. La memoria está diseñada para un funcionamiento síncrono o asíncrono. Actualmente funciona de forma síncrona, pero dispone de la señal MRdy para en un futuro trabajar de forma asíncrona. El circuito de selección

permite indicar qué porción de la palabra de memoria es la que se desea (un byte, dos bytes o una palabra completa de cuatro bytes).

Se dispone también de tres dispositivos de E/S: un teclado, una pantalla y un dispositivo genérico que se puede configurar para generar diversos tipos de interrupciones.

Finalmente, la Unidad de Control genera las señales de control para cada ciclo de reloj. La figura 2-8 muestra la Unidad de Control con mayor detalle. Se trata de una unidad de control microprogramada con secuenciamiento implícito. Las señales de control para el ciclo de reloj actual se almacenan en el registro de micro-instrucción (aquel con los campos A0, B, C, SelA, etc.). El contenido de este registro proviene de la memoria de control, concretamente del contenido en la posición a la que apunta el registro de micro-dirección. La micro-dirección almacenada en este registro puede modificarse usando el multiplexor "MUX A". Hay cuatro opciones: la microdirección actual más uno, una micro-dirección indicada en la propia micro-instrucción (que se solapa con SelA, SelB y parcialmente con SelE), la primera micro-dirección asociada al campo de código de operación de la instrucción del registro IR, y finalmente el valor cero, que es la dirección de la memoria de control donde se almacena la microrrutina correspondiente al fetch. La microdirección se puede seleccionar de forma condicional, para ello se usa el multiplexor "MUX C" que permite seleccionar los bits del registro de estado (SR) o valores de las señales de control de E/S.

Para generar los valores correspondientes a las señales selectoras del banco de registros RA, RB y RE se usan los circuitos selectores SelRA, SelRB y SelRE. Estos selectores toman como entrada los 32 bits del registro de instrucción (IR) por un lado y el campo SelA, SelB y SelE por otro, de forma que toman SelX como el desplazamiento dentro del registro de instrucción (de 0 a 32) desde donde tomar los siguientes 5 bits correspondientes a las señales RA, RB o RE. Permiten por tanto seleccionar 5 bits consecutivos de los 32 bits del registro de instrucción.

El multiplexor MR permite indicar si las señales RA, RB y RE serán literalmente los valores almacenados en SelA, SelB y SelE (MR=1) o bien si SelA, SelB y SelE indican el desplazamiento dentro de la instrucción donde están los valores a utilizar para RA, RB y RE. Esto último permite que en la instrucción se pueda indicar los registros a usar como operandos en el banco de registros, en lugar de indicarlos desde la micro-instrucción.

Para la señal Cop (código de operación en la ALU) la señal MC permite tomar el valor SelCop de la micro-instrucción (MC=1) o bien los 4 bits menos significativos del registro de instrucción (MC=0), es decir IR3-IR0.

En [11] se puede encontrar una descripción más detallada del procesador descrito anteriormente.

Capítulo 3

Análisis

El objetivo principal de este capítulo, es describir el proyecto mediante la obtención y especificación de los requisitos del simulador, que puede proporcionar información suficiente para un análisis detallado que, por lo tanto, puede servir para continuar diseñando e implementando (Capítulos 4, *Diseño*; and 5, *Implementación y despliegue*) un software que cumpla con esos requisitos.

Con el fin de obtener los requisitos del sistema, el tutor ha desempeñado el papel del cliente en diferentes reuniones, mientras que el alumno ha desempeñado los roles de analista, diseñador, programador y probador.

La sección 3.1 resume brevemente la descripción del proyecto. La sección 3.2 especifica los requisitos del sistema, empezando con los requisitos de usuario y finalizando con los requisitos funcionales y no-funcionales. La sección 3.2.2 especifica los caso de uso del sistema. Finalmente, la sección 3.3 indica el conjunto de leyes y regulaciones para la gestión del software.

3.1 Descripción del proyecto

El objetivo de este proyecto es construir una herramienta que permita simular con realismo el comportamiento de un procesador basado en la arquitectura indicada por el el tutor del proyecto, de forma que sirva como única herramienta para los alumnos a lo largo de la asignatura Estructura de Computadores.

Los simuladores actuales para la enseñanza de Estructura de Computadores, están focalizados a una función concreta, como puede ser la simulación de cachés, la simulación de código ensamblador o la simulación de microcódigo entre otros, pero a la hora de unificar todas estas

funcionalidades en una única herramienta, existe un vacío que genera una pérdida de tiempo en el aprendizaje de cada herramienta y la no posibilidad de ver una simulación completa.

El reto al que se enfrente cualquier simulador educativo es el de ser capaz de simular fielmente el funcionamiento de un dispositivo permitiendo al estudiante poder observar con el mayor detalle posible el comportamiento éste.

El sistema que se propone debe ser capaz de simular con realismo el comportamiento del procesador, permitiendo la definición del juego de instrucciones a utilizar para el posterior desarrollo de código ensamblador y su ejecución en el simulador con un alto nivel de detalle en cada uno de los ciclos de ejecución. De esta forma, los estudiantes podrán comprender fácilmente los contenidos teóricos expuestos en la asignatura y serán capaces de realizar todas las prácticas en una misma herramienta, pudiendo ver de forma incremental el desarrollo de software de bajo nivel sobre un procesador elemental.

3.2 Requisitos

Esta sección proporciona una descripción detallada de los requisitos de la aplicación. Para la tarea de la especificación de requisitos, se han seguido las prácticas recomendadas por IEEE [12]. De acuerdo con estas prácticas, una buena especificación debe abordar la funcionalidad del software, los problemas de rendimiento, las interfaces externas, otras características no funcionales y las limitaciones de diseño o implementación. Además, la especificación de los requisitos debe ser:

- **Completa:** el documento refleja todos los requisitos de software importantes.
- **Consistente:** los requisitos no deben generar conflictos entre sí.
- **Correcta:** cada requisito debe ser cumplido por el software según las necesidades del usuario.
- **Modificable:** la estructura de la especificación permite cambios en los requisitos de una manera simple, completa y consistente.
- **Clasificación basada en la importancia y la estabilidad:** cada requisito debe indicar su importancia y su estabilidad.

- **Trazable:** el origen de cada requisito es claro y se puede hacer referencia fácilmente en otras etapas.
- **Inequívoco:** cada requisito tiene una sola interpretación.
- **Verificable:** Cada requisito debe ser verificable, es decir, existe algún proceso para verificar que el software cumple con cada requisito.

A partir de los requisitos de los usuarios, que constituyen una referencia informal al comportamiento del producto que el cliente espera, derivamos los requisitos de software (en este caso, requisitos funcionales y no funcionales) que guiaron el proceso de diseño con información específica sobre la funcionalidad del sistema y otras características. Los requisitos recuperados se estructuraron de acuerdo con el siguiente esquema:

1. Requisitos de Usuario

- (a) **Capacidad:** el requisito describe la funcionalidad esperada del sistema como en casos de uso.
- (b) **Restricción:** el requisito especifica las restricciones o condiciones que el sistema debe cumplir.

2. Requisitos de Software

(a) Funcionales

- i. **Funcional:** el requisito describe la funcionalidad básica y el propósito del sistema mientras se minimiza la ambigüedad.
- ii. **Inverso:** el requisito limita la funcionalidad de la aplicación para aclarar su alcance.

(b) No-Funcionales

- i. **Rendimiento:** el requisito se relaciona con el rendimiento mínimo requerido del sistema resultante.
- ii. **Interfaz:** el requisito está relacionado con la interfaz de usuario de la aplicación.
- iii. **Escalabilidad:** el requisito está relacionado con la capacidad del sistema para adaptarse a cargas de trabajo cada vez mayores.

- iv. **Plataforma:** el requisito especifica las plataformas subyacentes de software y hardware en las que funcionará el sistema.

La tabla 3.1 proporciona la plantilla utilizada para la especificación de requisitos. Tenga en cuenta que para los requisitos del usuario, el formato de identificación será UR-XY, donde X indica el subtipo de requisito: requisitos de capacidad (C) o restricciones (R). YY corresponde al número de requisito en su subcategoría. Para los requisitos de software, se utilizará el formato de identificación SR-X-YYY, donde X indica si es un requisito funcional (F) o no funcional (NF), e Y representa su subcategoría: funcional (F), inversa (I), Rendimiento (P), interfaz (UI), escalabilidad (S) o plataforma (PL). ZZ corresponde al número de requisito en su subcategoría.

ID	Requisito ID.
Nombre	Nombre del requisito.
Tipo	Indica la categoría en la que se colocaría el requisito de acuerdo con el esquema descrito anteriormente.
Origen	Constituye la fuente del requisito. Puede ser el usuario, otro requisito u otros actores involucrados en el proyecto.
Prioridad	Indica la prioridad del requisito según su importancia. Un requisito puede ser identificado como <i>esencial</i> , <i>condicional</i> or <i>opcional</i> .
Estabilidad	Indica la variabilidad del requisito a través del proceso de desarrollo, definido como <i>estable</i> or <i>inestable</i> .
Descripción	Explicación detallada del requisito.

Tabla 3.1: *Plantilla para la especificación de requisitos.*

3.2.1 Requisitos de Usuario

Esta subsección especifica los requisitos de usuario.

ID	UR-C01
Nombre	Simulación del modelo hardware propuesto
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta simulará el comportamiento de la arquitectura definida para la asignatura Estructura de Computadores.

Tabla 3.2: *Requisito de usuario UR-C01.*

ID	UR-C02
Nombre	Definición de juego de instrucciones del simulador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la definición del juego de instrucciones a utilizar por el usuario siguiendo el formato especificado por el tutor.

Tabla 3.3: *Requisito de usuario UR-C02.*

ID	UR-C03
Nombre	Operaciones con el juego de instrucciones
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario las siguientes operaciones con el juego de instrucciones: cargar desde fichero, exportar a un fichero y generar el firmware.

Tabla 3.4: *Requisito de usuario UR-C03.*

ID	UR-C04
Nombre	Definición de juego del código ensamblador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la definición del código ensamblador a simular siguiendo el formato utilizado en la asignatura Estructura de Computadores y el juego de instrucciones cargado previamente.

Tabla 3.5: *Requisito de usuario UR-C04.*

ID	UR-C05
Nombre	Operaciones con el código ensamblador
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario las siguientes operaciones con el código ensamblador: cargar desde fichero, exportar a un fichero y generar el binario asociado.

Tabla 3.6: *Requisito de usuario UR-C05.*

ID	UR-C06
Nombre	Tipos de simulación
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir tres tipos diferentes de simulación: simulación ciclo a ciclo de reloj, simulación instrucción a instrucción y simulación completa del código.

Tabla 3.7: *Requisito de usuario UR-C06.*

ID	UR-C07
Nombre	Configuración de la velocidad de las simulaciones
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario seleccionar la velocidad de la simulación.

Tabla 3.8: *Requisito de usuario UR-C07.*

ID	UR-C08
Nombre	Esquemas de la arquitectura
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe mostrar la arquitectura de la CPU y la Unidad de Control que forman el simulador.

Tabla 3.9: *Requisito de usuario UR-C08.*

ID	UR-C09
Nombre	Información a mostrar
Tipo	Capacidad
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe mostrar al usuario la información del estado del simulador en cada paso de la simulación.

Tabla 3.10: *Requisito de usuario UR-C09.*

ID	UR-R01
Nombre	Aplicación web
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser una aplicación web.

Tabla 3.11: *Requisito de usuario UR-R01.*

ID	UR-R02
Nombre	Navegadores web
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de funcionar en los siguiente navegadores web: Microsoft Edge 30+, Mozilla Firefox 45+, Google Chrome 50+ y Safari 10+.

Tabla 3.12: *Requisito de usuario UR-R02.*

ID	UR-R03
Nombre	Interfaz de usuario
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario debe de ser compatible para uso en PCs y smartphones.

Tabla 3.13: *Requisito de usuario UR-R03.*

ID	UR-R04
Nombre	Tiempo por ciclo de reloj
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo de ejecución en media de un ciclo de reloj del simulador no debe de sobrepasar 0,1 segundos.

Tabla 3.14: *Requisito de usuario UR-R04.*

ID	UR-R05
Nombre	Ejecución en local
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	Las simulaciones serán realizadas en la máquina del usuario, siendo únicamente necesario el servidor para el acceso a la herramienta.

Tabla 3.15: *Requisito de usuario UR-R05.*

ID	UR-R06
Nombre	Conexión a internet
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta podrá ser ejecutada sin conexión a internet.

Tabla 3.16: *Requisito de usuario UR-R06.*

ID	UR-R07
Nombre	Tecnologías de desarrollo
Tipo	Restricción
Origen	Usuario
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser desarrollado mediante el lenguaje de programación HTML5.

Tabla 3.17: *Requisito de usuario UR-R07.*

3.2.2 Modelo de Casos de Uso

Un caso de uso representa un uso típico que realizará alguno de los futuros usuarios del sistema desarrollado. El diagrama que se muestra a continuación representa los casos de uso de un usuario que utilice esta herramienta desarrollada. Se ha partido el diagrama en dos para hacerlo mucho más legible. A continuación se muestra la primera parte del diagrama. En la página siguiente se mostrará la segunda parte.

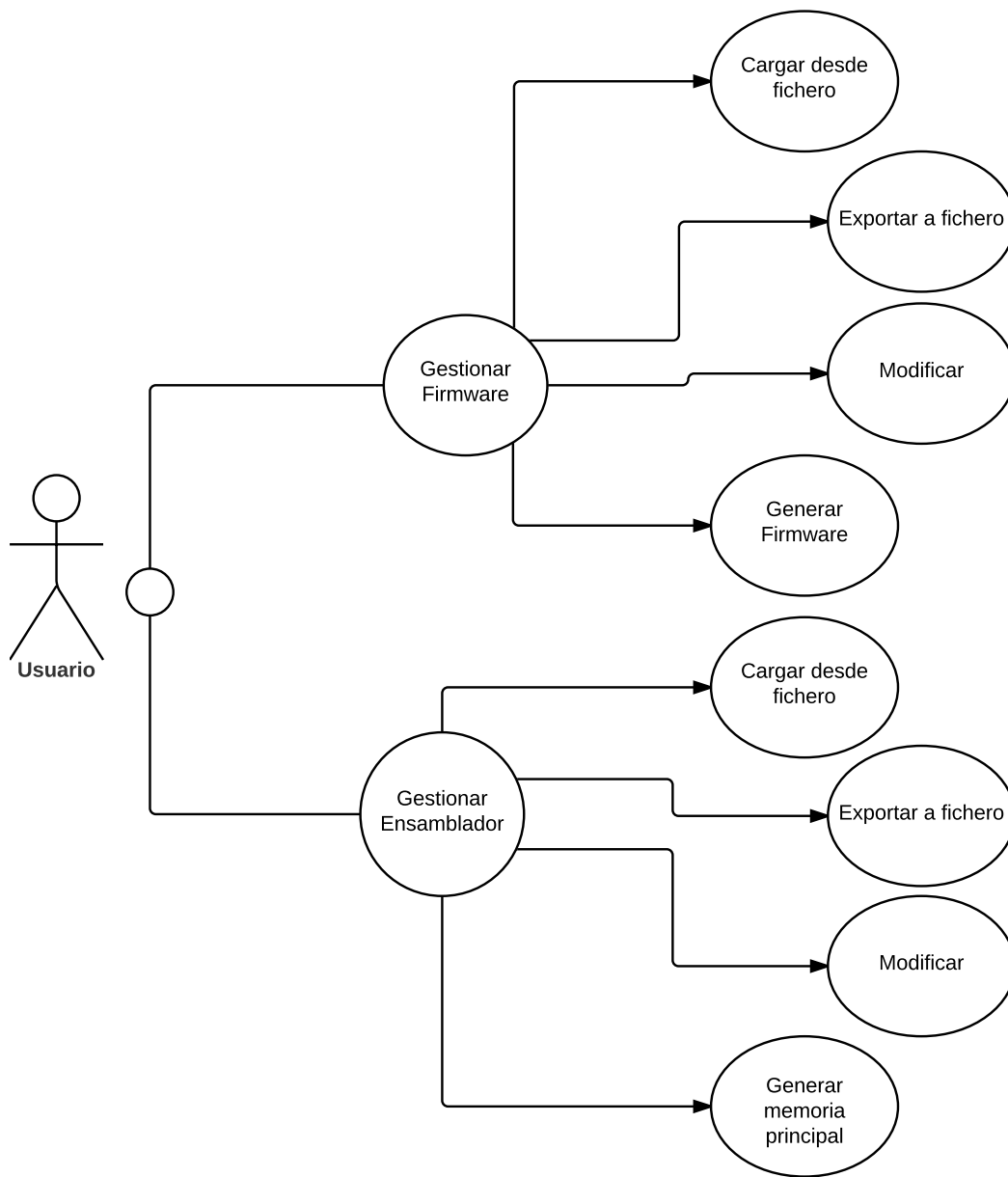


Figura 3-1: Diagrama Modelo Casos de Uso 1.

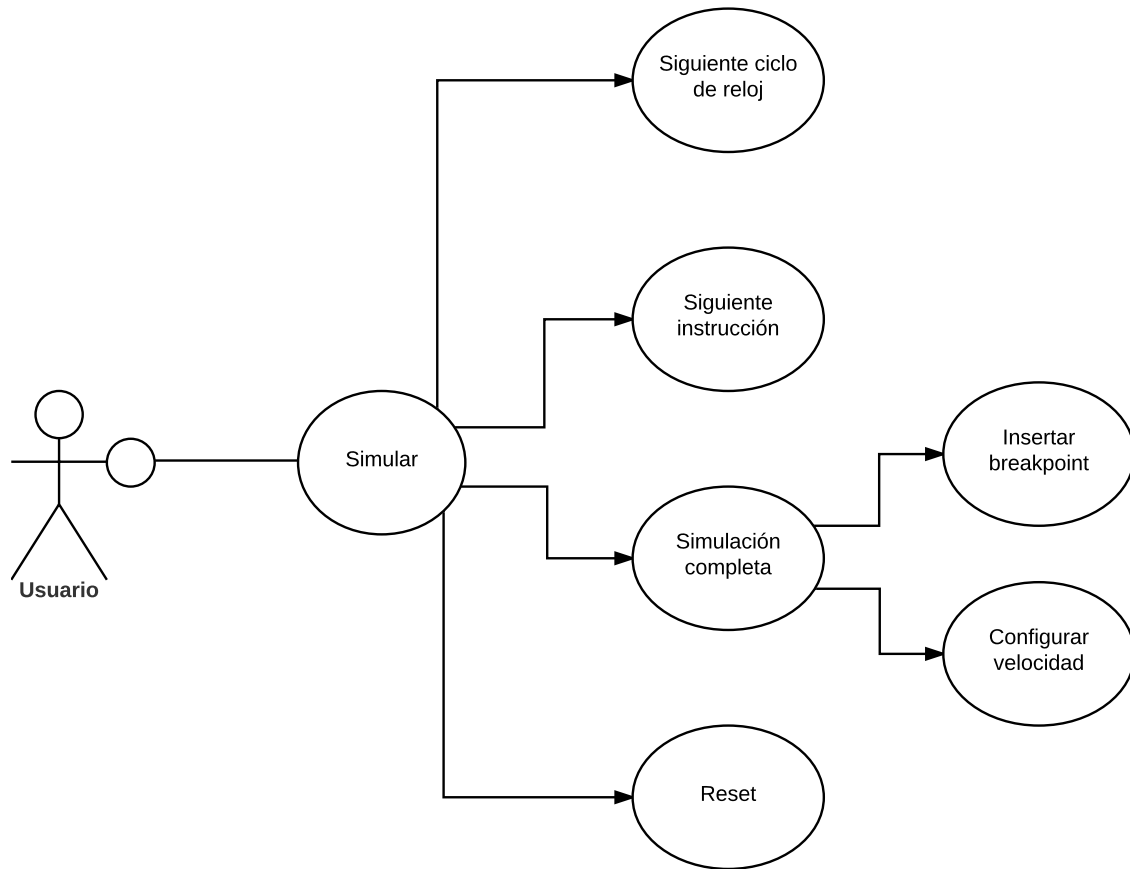


Figura 3-2: Diagrama Modelo Casos de Uso 2.

Para realizar la especificación clara, completa y detallada de cada uno de los casos de uso recogidos en el diagrama anterior se utilizará una tabla donde se recogerán los siguientes campos de información.

La tabla 3.18 proporciona la plantilla utilizada para la especificación de casos de uso. Tenga en cuenta que el formato de identificación será UC-XX, donde XX corresponde al número de caso de uso.

A continuación, se exponen los casos de uso.

ID	Caso de Uso ID.
Nombre	Nombre del Caso de Uso.
Actores	Describe el actor o los actores que intervienen en la realización del caso de uso.
Objetivo	Describe textualmente el cometido concreto del caso de uso.
Precondiciones	Muestra el estado del sistema que debe darse para que se pueda realizar el caso de uso.
Postcondiciones	Presenta el estado del sistema tras la realización del caso de uso.
Escenario Básico	Especifica la secuencia de pasos principales que se efectúan para realizar el caso de uso.

Tabla 3.18: *Plantilla de caso de uso.*

ID	UC-01
Nombre	Cargar juego de instrucciones
Actores	Usuario
Objetivo	Cargar en la herramienta un nuevo juego de instrucciones desde un fichero que indica el usuario.
Precondiciones	Ninguna.
Postcondiciones	El juego de instrucciones se carga en la herramienta, mostrándose en el editor de texto correspondiente al firmware.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la pestaña firmware. ■ Pulsa el botón cargar firmware. ■ Selecciona el fichero deseado y acepta.

Tabla 3.19: *Caso de Uso UC-01.*

ID	UC-02
Nombre	Exportar juego de instrucciones
Actores	Usuario
Objetivo	Exportar a un fichero el juego de instrucciones cargado en la memoria de control.
Precondiciones	La memoria de control ha sido generada.
Postcondiciones	Se genera un fichero en el dispositivo del usuario con el juego de instrucciones cargado previamente en la memoria de control.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña firmware. ▪ Pulsa el botón guardar firmware. ▪ Selecciona el nombre de fichero deseado y acepta.

Tabla 3.20: Caso de Uso UC-02

ID	UC-03
Nombre	Modificar juego de instrucciones
Actores	Usuario
Objetivo	Editar el juego de instrucciones desde la herramienta.
Precondiciones	Ninguna.
Postcondiciones	Las modificaciones realizadas por el usuario en el juego de instrucciones son mostradas en el editor de texto correspondiente al firmware.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña firmware. ▪ Realiza las modificaciones deseadas en el editor de texto.

Tabla 3.21: Caso de Uso UC-03.

ID	UC-04
Nombre	Generar firmware
Actores	Usuario
Objetivo	Generar la memoria de control a partir del juego de instrucciones definido en la herramienta.
Precondiciones	El juego de instrucciones debe estar definido en la herramienta.
Postcondiciones	Se genera la memoria de control asociada al juego de instrucciones definido por el usuario.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña firmware. ▪ Define el juego de instrucciones deseado en la herramienta. ▪ Pulsa el botón generar firmware.

Tabla 3.22: Caso de Uso UC-04.

ID	UC-05
Nombre	Cargar ensamblador
Actores	Usuario
Objetivo	Cargar en la herramienta un nuevo código ensamblador desde un fichero que indica el usuario.
Precondiciones	Ninguna.
Postcondiciones	El código ensamblador se carga en la herramienta, mostrándose en el editor de texto correspondiente al ensamblador.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña ensamblador. ▪ Pulsa el botón cargar ensamblador. ▪ Selecciona el fichero deseado y acepta.

Tabla 3.23: Caso de Uso UC-05.

ID	UC-06
Nombre	Exportar ensamblador
Actores	Usuario
Objetivo	Exportar a un fichero el código ensamblador cargado para la simulación.
Precondiciones	El código ensamblador ha sido cargado en el simulador.
Postcondiciones	Se genera un fichero en el dispositivo del usuario con el código ensamblador cargado previamente en el simulador.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la pestaña ensamblador. ■ Pulsa el botón guardar ensamblador. ■ Selecciona el nombre de fichero deseado y acepta.

Tabla 3.24: Caso de Uso UC-06.

ID	UC-07
Nombre	Modificar ensamblador
Actores	Usuario
Objetivo	Editar el código ensamblador desde la herramienta.
Precondiciones	Ninguna.
Postcondiciones	Las modificaciones realizadas por el usuario en el juego de instrucciones son mostradas en el editor de texto correspondiente al firmware.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la pestaña ensamblador. ■ Realiza las modificaciones deseadas en el editor de texto.

Tabla 3.25: Caso de Uso UC-07.

ID	UC-08
Nombre	Generar memoria principal
Actores	Usuario
Objetivo	Cargar el código ensamblador definido por el usuario en el simulador, generando el contenido de la memoria principal correspondiente al ensamblador.
Precondiciones	El código ensamblador debe estar definido en la herramienta.
Postcondiciones	Se genera la memoria principal asociada al código ensamblador definido por el usuario.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la pestaña ensamblador. ■ Define el código ensamblador deseado en la herramienta. ■ Pulsa el botón compilar.

Tabla 3.26: Caso de Uso UC-08.

ID	UC-09
Nombre	Siguiente ciclo de reloj
Actores	Usuario
Objetivo	Avanzar un ciclo de reloj en la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	La ejecución de la simulación avanza un ciclo de reloj, siendo modificado el estado del simulador.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la simulador. ■ Pulsa el botón siguiente micro-instrucción.

Tabla 3.27: Caso de Uso UC-09.

ID	UC-10
Nombre	Siguiente instrucción
Actores	Usuario
Objetivo	Avanzar una instrucción en la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	La ejecución de la simulación avanza una instrucción, siendo modificado el estado del simulador.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la simulador. ■ Pulsa el botón siguiente instrucción.

Tabla 3.28: Caso de Uso UC-10.

ID	UC-11
Nombre	Insertar breakpoint
Actores	Usuario
Objetivo	Insertar un punto de detención en una instrucción del código ensamblador a simular.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	Se añade un breakpoint en la simulación.
Escenario Básico	<ul style="list-style-type: none"> ■ El usuario selecciona la pestaña simulador. ■ Selecciona la pestaña Assembly Debugger. ■ Pulsa sobre la instrucción en la que añadir el breakpoint.

Tabla 3.29: Caso de Uso UC-11.

ID	UC-12
Nombre	Configurar velocidad
Actores	Usuario
Objetivo	Configurar la velocidad de la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	Se modifica la velocidad de la simulación.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña simulador. ▪ Modifica la velocidad mediante la barra deslizable de velocidad.

Tabla 3.30: Caso de Uso UC-12.

ID	UC-13
Nombre	Reinicio del simulador
Actores	Usuario
Objetivo	Reiniciar la simulación.
Precondiciones	La memoria de control y la memoria principal deben de estar generadas.
Postcondiciones	El simulador queda reiniciado y preparado para el comienzo de la simulación.
Escenario Básico	<ul style="list-style-type: none"> ▪ El usuario selecciona la pestaña simulador. ▪ Pulsa sobre el botón reset.

Tabla 3.31: Caso de Uso UC-13.

3.2.3 Requisitos Funcionales

Esta subsección especifica los requisitos funcionales.

ID	SR-F-F01
Nombre	Arquitectura de 32 bits
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe simular una arquitectura de 32 bits, implicando un banco de 32 registros, memoria principal con tamaño de 2^{32} bits y registros y buses de 32 bits.

Tabla 3.32: *Requisito Funcional SR-F-F01.*

ID	SR-F-F02
Nombre	Unidad de control microprogramable
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La unidad de control debe de ser microprogramable, con secuenciamiento implícito, saltos a nivel de microdirección y microsaltos condicionales.

Tabla 3.33: *Requisito Funcional SR-F-F02.*

ID	SR-F-F03
Nombre	Datos en complemento a2
Tipo	Funcional
Origen	UR-C01
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta operará con una ALU de números enteros en complemento a2.

Tabla 3.34: *Requisito Funcional SR-F-F03.*

ID	SR-F-F04
Nombre	Definición de juego de instrucciones
Tipo	Funcional
Origen	UR-C02
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario la definición del juego de instrucciones a utilizar.

Tabla 3.35: *Requisito Funcional SR-F-F04.*

ID	SR-F-F05
Nombre	Formato del juego de instrucciones
Tipo	Funcional
Origen	UR-C02
Prioridad	Esencial
Estabilidad	Estable
Descripción	El juego de instrucciones debe seguir el formato utilizado en la asignatura Estructura de Computadores, definiendo instrucciones del lenguaje ensamblador, su microcódigo asociado, nombrado de los registros y pseudoinstrucciones.

Tabla 3.36: *Requisito Funcional SR-F-F05.*

ID	SR-F-F06
Nombre	Modificación del juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la modificación del juego de instrucciones a utilizar por el usuario.

Tabla 3.37: *Requisito Funcional SR-F-F06.*

ID	SR-F-F07
Nombre	Carga de juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la carga de un fichero que contenga el juego de instrucciones.

Tabla 3.38: *Requisito Funcional SR-F-F07.*

ID	SR-F-F08
Nombre	Exportar juego de instrucciones
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir exportar el juego de instrucciones en un fichero.

Tabla 3.39: *Requisito Funcional SR-F-F08.*

ID	SR-F-F09
Nombre	Generación firmware del simulador
Tipo	Funcional
Origen	UR-C03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de generar el firmware simulador a partir del juego de instrucciones definido y cargado por el usuario.

Tabla 3.40: *Requisito Funcional SR-F-F09.*

ID	SR-F-F10
Nombre	Definición del código ensamblador
Tipo	Funcional
Origen	UR-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir al usuario la definición del código ensamblador a utilizar en la simulación.

Tabla 3.41: *Requisito Funcional SR-F-F10.*

ID	SR-F-F11
Nombre	Formato del código ensamblador
Tipo	Funcional
Origen	UR-C04
Prioridad	Esencial
Estabilidad	Estable
Descripción	El código ensamblador a simular debe de seguir el formato utilizado en la asignatura Estructura de Computadores, utilizando el lenguaje definido en el juego de instrucciones.

Tabla 3.42: *Requisito Funcional SR-F-F11.*

ID	SR-F-F12
Nombre	Edición del código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir modificar el código ensamblador.

Tabla 3.43: *Requisito Funcional SR-F-F12.*

ID	SR-F-F13
Nombre	Carga del código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir la carga de un fichero que contenga el código ensamblador.

Tabla 3.44: *Requisito Funcional SR-F-F13.*

ID	SR-F-F14
Nombre	Exportar código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe permitir exportar a un fichero el código ensamblador cargado en la herramienta.

Tabla 3.45: *Requisito Funcional SR-F-F14.*

ID	SR-F-F15
Nombre	Generación ejecutable del código ensamblador
Tipo	Funcional
Origen	UR-C05
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de generar el binario ejecutable asociado al código ensamblador definido para la simulación.

Tabla 3.46: *Requisito Funcional SR-F-F15.*

ID	SR-F-F16
Nombre	Simulación ciclo a ciclo de reloj
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación ciclo a ciclo de reloj del código ensamblador cargado por el usuario.

Tabla 3.47: *Requisito Funcional SR-F-F16.*

ID	SR-F-F17
Nombre	Simulación instrucción a instrucción
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación instrucción a instrucción del código ensamblador cargado por el usuario.

Tabla 3.48: *Requisito Funcional SR-F-F17.*

ID	SR-F-F18
Nombre	Simulación completa
Tipo	Funcional
Origen	UR-C06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir la simulación completa del código ensamblador cargado por el usuario.

Tabla 3.49: *Requisito Funcional SR-F-F18.*

ID	SR-F-F19
Nombre	Velocidad de simulación
Tipo	Funcional
Origen	UR-C07
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de permitir al usuario elegir la velocidad de simulación del código ensamblador cargado.

Tabla 3.50: *Requisito Funcional SR-F-F19.*

ID	SR-F-F20
Nombre	Esquema CPU
Tipo	Funcional
Origen	UR-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar el esquema de la CPU del simulador, modificando el color de las señales activas y buses de datos actualizados en cada ciclo de reloj.

Tabla 3.51: *Requisito Funcional SR-F-F20.*

ID	SR-F-F21
Nombre	Esquema Unidad de Control
Tipo	Funcional
Origen	UR-C08
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar el esquema de la Unidad de Control del simulador, modificando el color de las señales activas y buses de datos actualizados en cada ciclo de reloj.

Tabla 3.52: *Requisito Funcional SR-F-F21.*

ID	SR-F-F22
Nombre	Información de la simulación
Tipo	Funcional
Origen	UR-C09
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe de mostrar durante la ejecución el estado de los registros del simulador.

Tabla 3.53: *Requisito Funcional SR-F-F22.*

3.2.4 Requisitos No-Funcionales

Esta subsección especifica los requisitos no-funcionales.

ID	SR-NF-PL01
Nombre	Aplicación web
Tipo	Interfaz
Origen	UR-R01
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe de ser diseñado como una aplicación web.

Tabla 3.54: *Requisito Funcional SR-NF-PL01.*

ID	SR-NF-PL02
Nombre	Navegadores web
Tipo	Interfaz
Origen	UR-R02
Prioridad	Esencial
Estabilidad	Estable
Descripción	El simulador debe poder ser ejecutado en los navegadores: Microsoft Edge 30+, Mozilla Firefox 45+, Google Chrome 50+ y Safari 10+.

Tabla 3.55: *Requisito Funcional SR-NF-PL02.*

ID	SR-NF-PL03
Nombre	Plataforma de ejecución
Tipo	Interfaz
Origen	UR-R05
Prioridad	Esencial
Estabilidad	Estable
Descripción	Las ejecuciones de la herramienta deben ser realizadas en el dispositivo del usuario.

Tabla 3.56: *Requisito Funcional SR-NF-PL03.*

ID	SR-NF-PL04
Nombre	Internet
Tipo	Interfaz
Origen	UR-R06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta podrá ser ejecutada sin conexión a internet.

Tabla 3.57: *Requisito Funcional SR-NF-PL04.*

ID	SR-NF-PL05
Nombre	Lenguaje de programación HTML5
Tipo	Interfaz
Origen	UR-R06
Prioridad	Esencial
Estabilidad	Estable
Descripción	La herramienta debe ser desarrollada en el lenguaje de programación HTML5 (HTML + CSS + JavaScript).

Tabla 3.58: *Requisito Funcional SR-NF-PL05.*

ID	SR-NF-UI01
Nombre	Interfaz de usuario
Tipo	Interfaz
Origen	UR-R03
Prioridad	Esencial
Estabilidad	Estable
Descripción	La interfaz de usuario del simulador debe de ser compatible tanto con PCs como con plataformas móviles (tablets, smartphones, etc).

Tabla 3.59: *Requisito Funcional SR-NF-UI01.*

ID	SR-NF-P01
Nombre	Tiempo medio por ciclo de reloj
Tipo	Interfaz
Origen	UR-R03
Prioridad	Esencial
Estabilidad	Estable
Descripción	El tiempo medio de ejecución de un ciclo de reloj no debe exceder 0,1 segundos.

Tabla 3.60: *Requisito Funcional SR-NF-P01.*

3.3 Marco Regulador

Esta sección discute las restricciones necesarias teniendo en cuenta el marco regulador. En concreto, se especifican las restricciones legales aplicables al simulador.

3.3.1 Aspectos Legales

Para el uso de la mayoría de aplicaciones web, los usuarios deben de registrarse, y las bases de datos de las mismas manejan información confidencial de los usuarios, por lo que es necesario garantizar que terceros no puedan acceder a esa información. Una solución es cifrar la información transmitida mediante algún protocolo criptografico. En España, este requisito es especificado en el artículo 104 del RD 1720/2007 [13], que se ocupa de la Ley Española de Protección de Datos.

En contraste, la aplicación desarrollada no utiliza datos privados de los usuarios y tampoco transmite información confidencial a terceros, ya que es un simulador que únicamente utiliza los códigos generados por el usuario para ejecutarlos de forma local en la máquina del usuario.

Por otro lado, es crucial que nuestro simulador esté disponible como un software de código abierto. Queremos que sea tal que cualquiera pueda redistribuir el código o modificarlo por los términos de la Licencia Pública General Menor de GNU (LGPL) [14]. Para ello, nuestro simulador está disponible en el siguiente sitio web:

<https://www.arcos.inf.uc3m.es/wepsim/>.

Capítulo 4

Diseño

En este capítulo se realiza una descripción completa del simulador desarrollado, incluyendo la arquitectura interna y los diferentes componentes software, que componen la herramienta descritos con anterioridad en [15].

La sección 4.1 discute el estudio de la solución final, considerando las diferentes alternativas existentes para el diseño y desarrollo del simulador. En la sección 4.1.1 se indica la solución elegida y la compara con las alternativas consideradas. La sección 4.2 describe cada uno de los componentes que componen el simulador.

4.1 Estudio de la solución final

Existen diferentes modelos arquitectónicos a la hora de realizar el diseño de una aplicación web. La primera opción, es el modelo cliente-servidor en el cual las tareas se reparten entre dos roles: un proveedor que proporciona recursos o servicios que es llamado servidor, y un consumidor que contacta con el servidor con el objetivo de hacer uso de los recursos que este provee. Es un modelo distribuido en el que la lógica de negocio y el cómputo recae en el servidor, y el cliente provee la interfaz de usuario, realizando validaciones y procesos menores de la lógica de negocio. La segunda opción, es un modelo en el que toda la aplicación es ejecutada en el cliente, siendo únicamente necesario el servidor para el acceso al código de la aplicación. Este modelo, posibilita incluso que el servidor no sea necesario en el caso de existir el código de la aplicación web en el cliente, pudiendo ser ejecutada sin la necesidad de internet. Ambos modelos, poseen una alta compatibilidad al hacer que las aplicaciones no sean dependientes del sistema operativo del usuario, sino del navegador web que este utiliza.

Debido a los requisitos de usuario establecidos en este proyecto (Sección 3.2.1), la segunda opción es la que cumple con la necesidad de no necesitar acceso a internet, de forma que la funcionalidad total del simulador quede en el dispositivo del usuario.

Una vez decidido el modelo de diseño, es necesario tener en cuenta las diferentes tecnologías a utilizar. Como se indicó en (Sección 3.2.1), es necesario que la aplicación sea diseñada siguiendo el lenguaje de programación HTML5, que está formado por HTML, CSS y JavaScript. De este modo, es necesario contemplar los diferentes frameworks y bibliotecas existentes que ayudan a la hora de realizar el diseño de la aplicación, posibilitando el funcionamiento de esta tanto en ordenadores como en dispositivos móviles. Estas tecnologías descritas anteriormente en el Apartado 2.2, proporcionan distintas funcionalidades y enfoques a la hora de diseñar la aplicación, permitiendo la compatibilidad de la aplicación con la amplia mayoría de los navegadores web. En la Tabla 4.1, se analizan las funcionalidades y compatibilidades que proporcionan estas tecnologías, valorando si cumplen con los requisitos establecidos en este proyecto.

Simulador	SPIM	MARS	PC88110	WebMIPS	WepSIM	P8080E	MicMac
Ensamblador	✓	✓	✓	✓	✓		
Microprogramación					✓	✓	✓
Multiplataforma	✓			✓	✓		
Interactivo					✓		
Juego de instrucciones personalizable					✓		

Tabla 4.1: Comparación de frameworks y bibliotecas.

De este modo, se puede observar como pese a que *Angular.js* nos proporciona la mayoría de funcionalidades requeridas, tiene una curva de aprendizaje bastante mayor que *jQuery*, lo cual hace que *jQuery* se imponga a la hora de su elección debido a que la ganancia de uso que proporcionaría *Angular.js* es inferior a este inconveniente. Por otro lado, sería interesante el uso de *Dojo* a la hora de utilizar una tecnología que nos proporcione funcionalidad y efectos en la interfaz de usuario y nos ayude a la captación de eventos, pero queda descartado al estar enfocado a aplicaciones que utilizan *AJAX*. De esta forma, *Bootstrap* será la tecnología elegida para dotar a la aplicación de la funcionalidad requerida a nivel de interfaz de usuario y complementando tanto a JavaScript como a jQuery.

4.1.1 Solución elegida

Para que los profesores de la asignatura Estructura de Computadores puedan hacer uso de una herramienta que sirva de ayuda para la explicación de los conceptos teóricos de la asignatura, y los alumnos puedan utilizarla para comprender estos conceptos y realizar posteriormente las prácticas de la asignatura, se propone el diseño e implementación de una herramienta web que simule con realismo en funcionamiento de un procesador elemental con unidad de control microprogramable.

Este simulador, será desarrollado como una herramienta web debido a la portabilidad que proporciona, ya que podrá ser ejecutado sobre un gran número de diferentes dispositivos independientemente del sistema operativo que utilice, puesto que únicamente necesita un navegador web para su correcto funcionamiento. De esta forma, los profesores y alumnos podrán hacer uso de la herramienta sin depender de su instalación en el dispositivo a utilizar, incluso pudiendo los alumnos realizar las prácticas sobre dispositivos móviles.

Para lograr dicha portabilidad, el simulador ha sido desarrollado en HTML5 (HTML + JavaScript + CSS) haciendo posible su ejecución en cualquier plataforma (smartphones, tablet, PC, etc.) que pueden ejecutar Microsoft Edge, Mozilla Firefox, Google Chrome o Safari. Además, la herramienta depende de los siguientes frameworks/bibliotecas: JQuery, JQueryUI, JQuery Mobile, Knockout y BootStrap.

Por tanto, la solución elegida es capaz de unificar en una misma herramienta todas las funcionalidades requeridas para la enseñanza de Estructura de computadores con un alto nivel de detalle, con alta disponibilidad al facilitarse su como una herramienta web, y con una gran portabilidad puesto que podrá ser ejecutada sobre un gran número de diversos dispositivos, buscando en todo momento una solución sin la necesidad de servidor.

4.2 Arquitectura de WepSIM

La arquitectura de la solución presentada en este trabajo consta de tres elementos principales:

- Modelo hardware: permite definir el hardware a usar.
- Modelo software: permite definir el juego de instrucciones a utilizar.

- Kernel de simulación: simula el funcionamiento del hardware ejecutando el microcódigo/lenguaje máquina definido con anterioridad.

El modelo hardware permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular. La forma de definir estos elementos equilibra dos objetivos contrapuestos: es suficientemente completa como para imitar los principales aspectos de la realidad, pero es lo suficientemente mínima para facilitar su uso. Ante todo se persigue que sea una herramienta didáctica.

El modelo software permite definir el microcódigo y el ensamblador basado en este microcódigo de la forma tan intuitiva posible. El ensamblador a usar viene dado por un conjunto de instrucciones que puede ser definido por el usuario e intenta ser lo suficientemente flexible como para poder definir diferentes tipos y juegos de instrucciones, como por ejemplo MIPS o ARM.

El tercer elemento de la arquitectura propuesta es un kernel que toma como entrada el modelo hardware descrito y el modelo software de trabajo, y se encarga de mostrar el funcionamiento del hardware con el software dado.

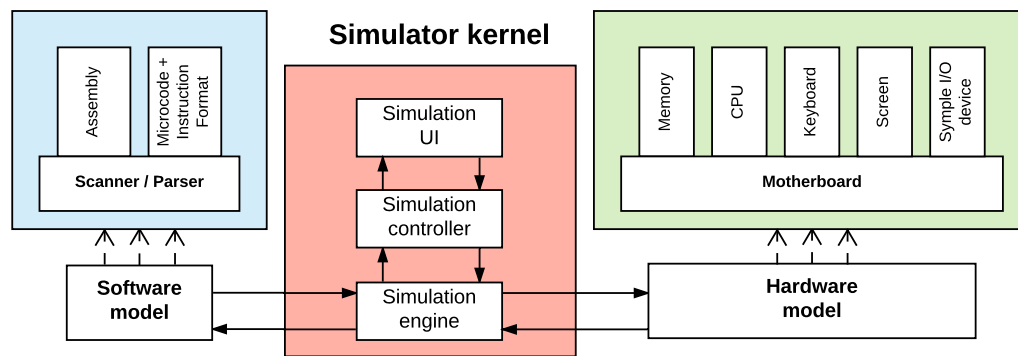


Figura 4-1: Arquitectura de WepSIM.

La figura 4-1 resume la arquitectura de WepSIM. El punto de inicio es el modelo hardware que describe el procesador a ser simulado. Ello incluye el procesador, la memoria y algunos dispositivos de E/S: teclado, pantalla y un dispositivo de E/S simple que genera interrupciones. El modelo hardware describe el estado global del procesador. A partir del estado global del procesador, el kernel de simulación actualiza el estado en cada ciclo de reloj.

La unidad de control simulada almacena las señales de control de cada ciclo en una memoria de control. La memoria de control tiene todos los microprogramas para las instrucciones con

las que trabaja el procesador, y el fetch para leer la instrucción de memoria y decodificarla.

El microcódigo (el contenido de la memoria de control) junto con el formato de cada instrucción (campos de la instrucción y su longitud) se describe en un fichero de texto. El modelo software lee este fichero, lo traduce a binario y lo carga en el procesador. La definición del lenguaje ensamblador a utilizar se describe junto con el microcódigo, y el modelo software permite traducir a binario programas escritos en dicho ensamblador.

El kernel de simulación pregunta al subsistema del modelo software por el microcódigo definido, la descripción del formato de instrucción y el contenido de la memoria principal. Los binarios se cargan en los elementos del modelo hardware, y a continuación el kernel de simulación actualiza el estado global en cada ciclo de reloj.

WepSIM dispone de un controlador de simulación que se encarga de actualizar el ciclo de reloj y mostrar el estado global. El subsistema de interfaz de simulación actualiza la interfaz de usuario. Cuando el usuario usa la interfaz de usuario para solicitar una operación, el subsistema de interfaz de simulación traslada la petición al controlador de simulación. Como se puede ver, se usa un Modelo- Vista-Controlador (MVC) básico para la arquitectura de WepSIM.

4.2.1 Modelo hardware

El modelo hardware que usa WepSIM permite definir los distintos elementos típicos de un computador (memoria principal, procesador, etc.) de una forma modular y de manera que sea posible añadir, quitar o modificar estos elementos.

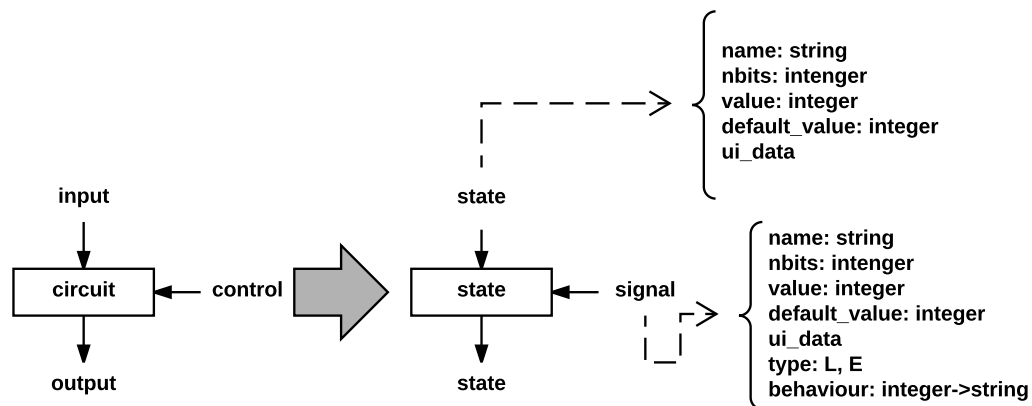


Figura 4-2: Modelado del hardware.

La figura 4-2 introduce el modelo propuesto. Cada elemento del circuito se describe como una caja negra con posibles entradas, posibles salidas y señales de control (que controlan las

posibles transformaciones de las entradas a las salidas). El subsistema del modelo hardware transforma esta caja negra en dos conjuntos de objetos: estados y señales. Un estado tiene un identificador (el nombre), el valor (un valor entero) y un valor inicial (el valor por defecto). Los valores que puede tomar son valores naturales dentro de un rango, dado por el número de bits con los que se representa el estado. Una señal es un estado especial que controla el valor de otros estados o señales. Hay dos atributos asociados a las señales (y no a los estados): el tipo de señal (por nivel o por flanco) y su comportamiento. Para cada valor de señal una cadena de caracteres describe en un Lenguaje Simple lo que la señal mueve o transforma. Este Lenguaje Simple se compone principalmente de instrucciones que representan las operaciones elementales.

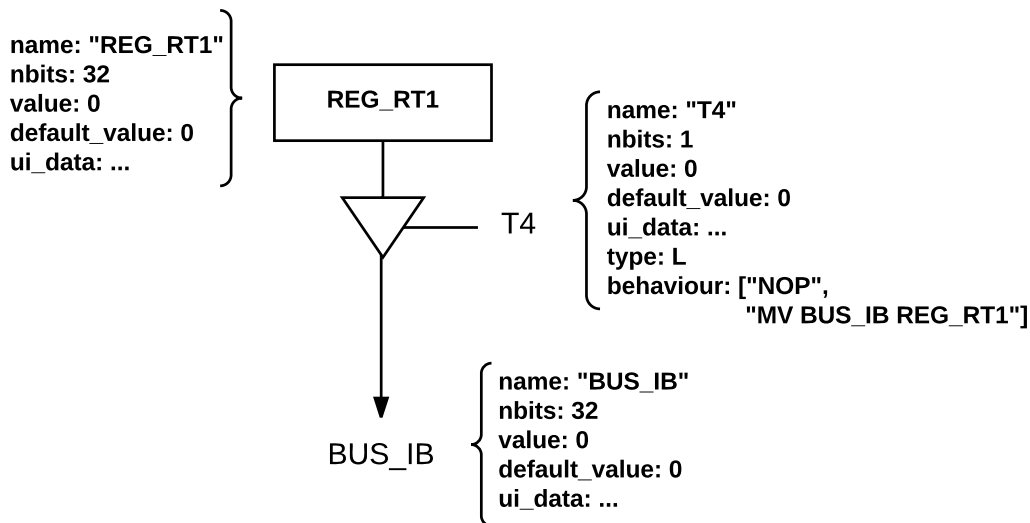


Figura 4-3: Ejemplo de modelado de una puerta triestado.

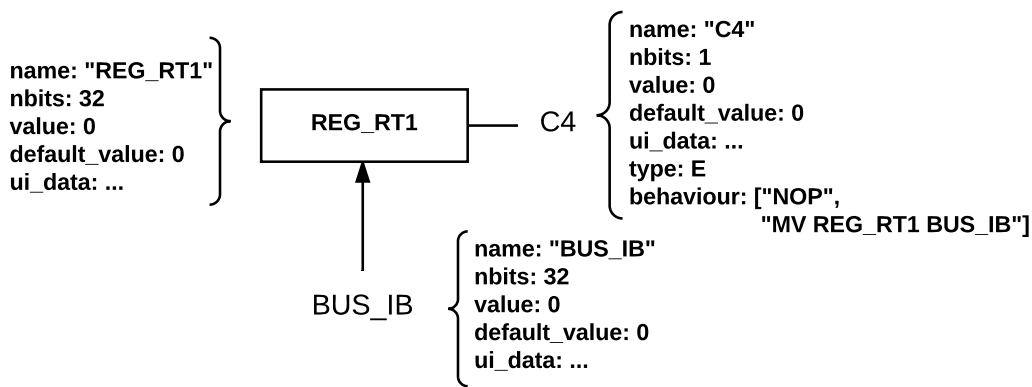


Figura 4-4: Ejemplo de modelado de un registro.

Las figuras 4-3 y 4-4 muestran dos ejemplos: un triestado y un registro. El triestado controla dos estados: el estado del bus al que se conecta BUS IB y el estado del registro de entrada, REG RT1 en este caso. Ambos representan el valor a la salida de la puerta (BUS IB) y el valor del registro RT1 (REG RT1). La señal T4 se encarga de indicar cuándo el valor del registro RT1 se envía a la salida. Esta señal T4 es una señal por nivel (tipo: L), con valor cero no tiene efecto (comportamiento "NOP"). Cuando el valor de la señal es uno entonces el comportamiento es el de copiar el valor del registro RT1 a la salida (comportamiento "MV BUS IB REG RT1").

El ejemplo con el registro (figura 4-4) es similar. En este caso trabaja con dos estados: el contenido del registro RT1 y el contenido situado a la entrada (BUS IB). La señal C4 controla cuándo se almacena en el registro RT1 el valor que hay en la entrada. La diferencia está en el tipo de señal: C4 es una señal por flanco de bajada (tipo: E), por lo que al final del ciclo de reloj (pasa de uno a cero) si la señal vale uno entonces el comportamiento es el de copiar el valor situado a la entrada al registro (comportamiento "MV REG RT1 BUS IB").

El Lenguaje Simple usado para definir los comportamientos añade a las operaciones elementales otras operaciones necesarias. Por ejemplo disparar una señal ("FIRE C4") que ayuda a propagar el efecto de una señal al reevaluar la señal inmediata que podría verse afectada. Otro ejemplo lo encontramos en dos operaciones que pueden ser muy útiles a la hora de depurar: imprimir el valor de un estado ("PRINT E BUS IB") e imprimir el valor de una señal ("PRINT S C4").

4.2.2 Modelo software

Una vez definido el procesador elemental usando el modelo hardware propuesto, toca describir el conjunto de instrucciones que es capaz de ejecutar así como el microcódigo que lo orquesta. En un fichero de texto se define el formato de las instrucciones máquina junto con el cronograma asociado a la ejecución de cada una de las instrucciones máquina. La figura 4-5 muestra un ejemplo de definición para la instrucción li (load immediate), que almacena un valor inmediato en un registro.

El fichero con el cronograma de fetch y todos los cronogramas de las instrucciones define el microcódigo para la plataforma WepSIM. El simulador permite la definición de diferentes juegos y formatos de instrucciones. Inicialmente se ha implementado un subconjunto de las

```

li reg val
{
    co=000010,
    nwords=1,
    reg=reg(25,21),
    val=inm(15,0),
    {
        (SE=0, OFFSET=0, SIZE=10000, SE=1, T3=1,
        LE=1, MR=0, SELE=10101, A0=1, B=1, C=0)
    }
}

```

Figura 4-5: Ejemplo de formato de instrucción.

instrucciones del MIPS, pero es posible definir instrucciones de otros conjuntos de forma similar. En este fichero se pueden asignar códigos simbólicos a los registros del banco de registros, lo que permite que en los programas escritos en ensamblador se puedan usar dichos símbolos (por ejemplo, registro \$t3 en la figura 4-6).

```

.text
main:  li  $t3 8
       li  $t5 10
       add $t6 $t3 $t5

```

Figura 4-6: Ejemplo de código fuente en ensamblador.

El campo “co” identifica el código de instrucción máquina, que es un número binario de 6 bits. Esto permite definir hasta 64 instrucciones distintas. Dado que los últimos 4 bits de la instrucción pueden usarse para seleccionar la operación en la ALU, es posible seleccionar hasta 16 operaciones aritmético-lógicas con un mismo código de instrucción, por lo que se podrían tener 79 (63+16) instrucciones en total.

Cuando WepSIM carga el microcódigo, cada código de instrucción tiene asociado una dirección de comienzo en la memoria de control donde se almacena el cronograma asociado. Esta tabla con dos columnas (el código de instrucción y su dirección de comienzo asociada en la memoria de control) se carga en la ROM co2microAddr mostrada en la figura 2-8.

El campo “nwords” define cuantas palabras precisa la instrucción para su definición y carga en memoria. Una palabra en WepSIM son 4 bytes.

Para cada campo de la instrucción se define el bit inicial, el bit final (ambos incluidos) y el tipo de campo (registro, valor inmediato, dirección absoluta y dirección relativa a PC). Una vez

definido el formato, se definen todas las microinstrucciones que necesita la instrucción máquina definida para su ejecución. Todas las microinstrucciones se encuentran encerradas entre llaves y cada microinstrucción está formada por una lista de tuplas (señal, valor) encerradas entre paréntesis. Para la instrucción definida en la figura 4-5 se precisa de una sola microinstrucción, en la que se indican qué señales se activan durante un ciclo de reloj. Para las señales no indicadas se asume que su valor es 0 durante el ciclo de reloj correspondiente.

Una vez cargado el microcódigo en WepSIM, es posible cargar cualquier fichero ensamblador que haya sido codificado usando las instrucciones máquina definidas anteriormente en el microcódigo.

En la figura 4-6 se muestra un ejemplo de código fuente en ensamblador que se puede usar en WepSIM. Este ejemplo en particular muestra un código estilo MIPS. Para que un programa en ensamblador pueda utilizar la instrucción de carga inmediata *li* (load immediate) y de suma *add* (addition), deben haber sido definidas previamente en el microcódigo. WepSIM puede comprobar los errores de sintaxis y construir el binario mediante el rellenado de los campos descritos en la definición del microcódigo correspondiente a la instrucción. La figura 4-7 muestra un ejemplo de traducción a binario para la instrucción *li \$2 5* en función del formato definido en la figura 4-5. También se debe haber definido en el fichero de microcódigo el valor del registro asociado a la etiqueta *\$2* (00100 en este caso).

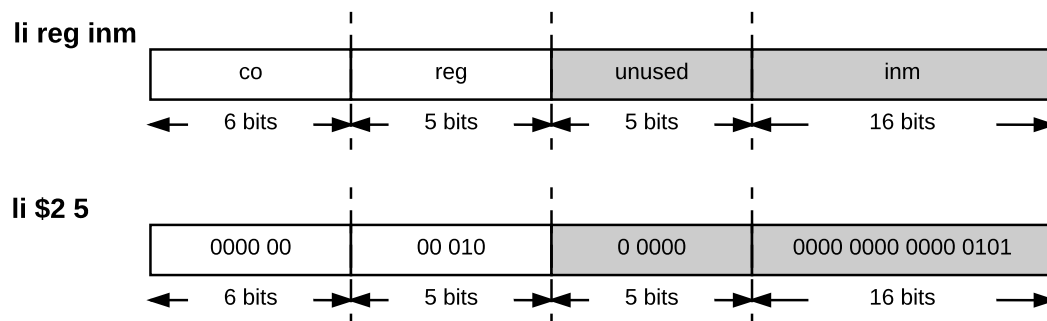


Figura 4-7: Formato de instrucción descrito en el microcódigo y ejemplo de su traducción en binario.

Una de las grandes ventajas del simulador WepSIM es que no está limitado a un conjunto de instrucciones concreto. Se puede definir un amplio conjunto de instrucciones de procesadores reales o inventados. Se puede usar para añadir por ejemplo, a un conjunto de instrucciones MIPS, otras instrucciones diferentes no incluidas en dicho conjunto de instrucciones.

4.2.3 Kernel del simulador

El kernel de simulación diseñado en WepSIM, es el componente que sirve de unión entre el modelo hardware y software. Su función principal reside en la simulación del computador utilizando la definición del modelo hardware y el juego de instrucciones definido, realizando la ejecución de los códigos ensamblador, siendo el controlador de la herramienta.

Además de realizar el proceso de simulación, este componente a su vez realiza la actualización de la interfaz de usuario del simulador, en donde se muestra el valor de cada uno de los registros de la máquina y el estado de los buses y señales que conforman el modelo hardware.

En una simulación, este kernel en primer lugar se encarga de importar en la memoria de control el resultado de la generación del juego de instrucciones definido por el usuario. Una vez cargado el lenguaje máquina, se encarga de cargar el resultado del código ensamblador compilado en la memoria principal del simulador. Una vez cargada esta información, a través de la información que le proporcionan tanto el modelo hardware como el modelo software, se encarga de gestionar los eventos que son generados debido a la interacción del usuario con el simulador a través de la interfaz de usuario. Estos eventos son generados por el módulo de la interfaz de usuario y son tratados por el kernel del simulador, identificando cada uno de los eventos y ejecutando la función asociada a estos eventos. Este diseño orientado a eventos mantiene el principio de diseño modular de la herramienta, de forma que en un futuro éste kernel pueda ser utilizado sin necesidad de una interfaz gráfica.

Capítulo 5

Implementación y despliegue

Este capítulo trata de la implementación y despliegue del software. En cuanto a la implementación del sistema, se explican las partes más complicadas del código en (Sección 5.1, *Implementación*). Por otro lado, explicamos los pasos necesarios para desplegar el sistema final (Sección 5.2, *Despliegue*)

5.1 Implementación

Como hemos explicado en el capítulo 3, *Análisis*, hemos implementado el simulador utilizando el lenguaje de programación JavaScript junto con HTML5, CSS y las bibliotecas/frameworks JQuery, JQueryUI, JQuery Mobile, Knockout y BootStrap. El motor de simulación es el encargado de ejecutar cada uno de los ciclos de reloj del simulador, tomando como entradas tanto el modelo hardware como el modelo software, pero el desarrollador ha debido de diseñar e implementar el algoritmo que posibilita esta ejecución.

Además, hemos trabajado en conseguir una herramienta que sea capaz de generar la memoria de control mediante la definición del juego de instrucciones por parte del usuario, y de generar el código binario asociado al código ensamblador definido por el usuario; el cual depende del juego de instrucciones definido previamente. Para ello, se han diseñado e implementado dos compiladores diferentes, capaces de generar los binarios correspondientes además de las estructuras de datos necesarias para ayudar al motor de simulación a lo largo de la ejecución.

De esta forma, en 5.1, podemos observar el pseudocódigo del proceso de generación de la memoria de control, mientras que en 5.2 podemos ver el proceso de compilación del código ensamblador definido por el usuario en función de la memoria de control previamente generada,

detallando el proceso de compilación del segmento de datos en 5.3 y el proceso de compilación del segmento de texto en 5.4. En 5.5 podemos ver el pseudocódigo de como se realiza una simulación, realizando la comprobación de segmentos de memoria, tipo de simulación, y ejecución del ciclo o instrucción correspondiente.

Pseudocode 5.1 Proceso de compilación del juego de instrucciones

```

1: function COMPILE_FIRMWARE(context )
2:   token = getToken();
3:   firmware = null;
4:   instructions = null;
5:   registers = null;
6:   pseudoinstructions = null;
7:   while (!is_registers(token) and token != null) do
8:     instruction = null;
9:     instructions[signature]=readSignature(getNextToken());
10:    instructions[co]=readCo(getNextToken());
11:    if (getNextToken() == COP) then
12:      instruction[cop]=readCop(getNextToken());
13:    end if
14:    while (is_Field(getNextToken())) do
15:      instructions[field].add=readField(getToken());
16:    end while
17:    while (is_microcode(getNextToken())) do
18:      instructions[microcode].add=readCycle(getToken());
19:    end while
20:    firmware.add(instructions);
21:  end while
22:  if (is_registers(getToken())) then
23:    registers = readRegisters(getToken());
24:  end if
25:  if (is_pseudoInstructions(getToken())) then
26:    pseudoinstructions = readPseudoinstructions(getToken());
27:  end if
28:  firmware.add(instructions);
29:  firmware.add(registers);
30:  firmware.add(pseudoinstructions);
31:  Return firmware;
32: end function

```

Pseudocode 5.2 Proceso de compilación de código ensamblador

```
1: function COMPILE_ASSEMBLY( context,datosCU)
2:   dataSegment = null;
3:   textSegment = null;
4:   mainMemory = null;
5:   if (!is_data_segment(getToken())) then
6:     Return;
7:   end if
8:   dataSegment = read_data_segment(context[dSegment]);
9:   if (!is_text_segment(getToken())) then
10:    Return;
11:  end if
12:  textSegment = read_text_segment(context[tSegment]);
13:  mainMemory.add(dataSegment);
14:  mainMemory.add(textSegment,dataSegment.labels);
15:  Return mainMemory;
16: end function
```

Pseudocode 5.3 Proceso de compilación de código ensamblador: segmento de datos

```

1: function READ_DATA_SEGMENT(context)
2:   labels = null;
3:   token = getToken();
4:   while (!is_End(getNextToken())) do                                ▷ Check Labels
5:     if (isLabel(getToken())) then
6:       labels.add(getLabel(getToken()));
7:     end if
8:   end while
9:   dataSegment[labels].add(labels);
10:  token = getFirstToken();
11:  for (i in context[directives]) do                                ▷ Read Directives
12:    possible_datatype = i;
13:    if (possible_datatype == ".byte") then dataSegment[directives].add(readByte(i));
14:    end if
15:    if (possible_datatype == ".half") then dataSegment[directives].add(readHalf(i));
16:    end if
17:    if (possible_datatype == ".word") then dataSegment[directives].add(readWord(i));
18:    end if
19:    if (possible_datatype == ".space") then dataSegment[directives].add(readSpace(i));
20:    end if
21:    if (possible_datatype == ".ascii") then dataSegment[directives].add(readAscii(i));
22:    end if
23:    if (possible_datatype == ".asciiz") then dataSegment[directives].add(i);
24:    end if
25:    if (possible_datatype == ".align") then dataSegment[directives].add(readAlign(i));
26:    end if
27:  end for
28:  Return dataSegment
29: end function

```

Pseudocode 5.4 Proceso de compilación de código ensamblador: segmento de texto

```

1: function READ_TEXT_SEGMENT(context, labels)
2:   for (i in labels do                                     ▷ Read Labels
3:     replaceLabel(context, i);                               ▷ Replace Label in Text Segment
4:   end for
5:   for (i in context[instructions]) do                     ▷ Read Instructions
6:     auxInstruction = fillFields(i);
7:     auxBinary = null;
8:     if (!isInstruction(auxInstruction[0])) then             ▷ Check if instruction exist
9:       Return;
10:    end if auxBinary = generateBinary(auxInstruction);
11:    if (auxBinary == null) then
12:      Return;
13:    end if textSegment.add(auxBinary);
14:  end for
15:  Return textSegment
16: end function

```

Pseudocode 5.5 Proceso de simulación

```

1: function RUN_SIMULATION( )
2:   if (!possible_execute()) then
3:     Return;
4:   end if
5:   change_play_button();
6:   execute_simulation_inchain();
7: end function
8: function EXECUTE_SIMULATION_INCHAIN( )
9:   if (end_code()) then
10:    Return;
11:  end if
12:  if (is_breakpoint()) then
13:    Return;
14:  end if
15:  if (execute_mode == microInstruction) then
16:    execute_microInstruction();
17:  end if
18:  if (execute_mode == instruction) then
19:    execute_instruction();
20:  end if
21: end function
22: function EXECUTE_MICROINSTRUCTION( )
23:   if (!possible_execute()) then
24:     Return;
25:   end if
26:   compute_general_behaviour(CLOCK);                                ▷ Execute Cycle
27:   update_UI();
28: end function
29: function EXECUTE_INSTRUCTION( )
30:   if (!possible_execute()) then
31:     Return;
32:   end if
33:   do
34:     compute_general_behaviour(CLOCK);                                ▷ Execute Cycle
35:   while (reg_microaddr!=0 and mem_control[microAddr]!=undefined)    ▷ Check Next Cycle
36:     update_UI();                                                    ▷ Update User Interface
37: end function

```

5.2 Despliegue

En esta sección se presenta el despliegue de la herramienta. Para ello, indicamos las especificaciones técnicas recomendadas para que el usuario final obtenga la mejor experiencia posible con la herramienta:

- **Sistema Operativo:** Ubuntu 16.04.2 LTS (Linux distribution) / Windows 10 / MacOS 10.12.5.
- **Procesador:** Intel(R) Core(TM) i3 CPU 6300 @3.8GHz or higher.
- **Random-Access Memory (RAM):** 4 GB or higher.
- **Almacenamiento:** 1 GB of free space in the Hard Disk Drive (recomendado para el navegador web).
- **Red:** La conexión a internet no es necesaria para la ejecución de la herramienta, únicamente para el acceso a ella.
- **Software:** Los siguientes navegadores web son los recomendados para el uso de la herramienta:
 1. Mozilla Firefox 45+.
 2. Google Chrome 50+.
 3. Microsoft Edge 30+.
 4. Safari 10+.

En caso de desear el usuario descargar el código fuente de la herramienta para realizar cualquier modificación en la definición del modelo hardware o cualquier otro módulo, es necesario explicar la estructura de ficheros que componen el simulador y las dependencias que existen entre sí. De esta forma, en 5-1 podemos ver los ficheros que componen la herramienta web, los cuales tienen una serie de dependencias indicadas en 5-2.

Los ficheros que componen WepSIM son detallados a continuación:

- **index.html:** este fichero se encarga de la vista de la herramienta, generando la interfaz de usuario de la herramienta.

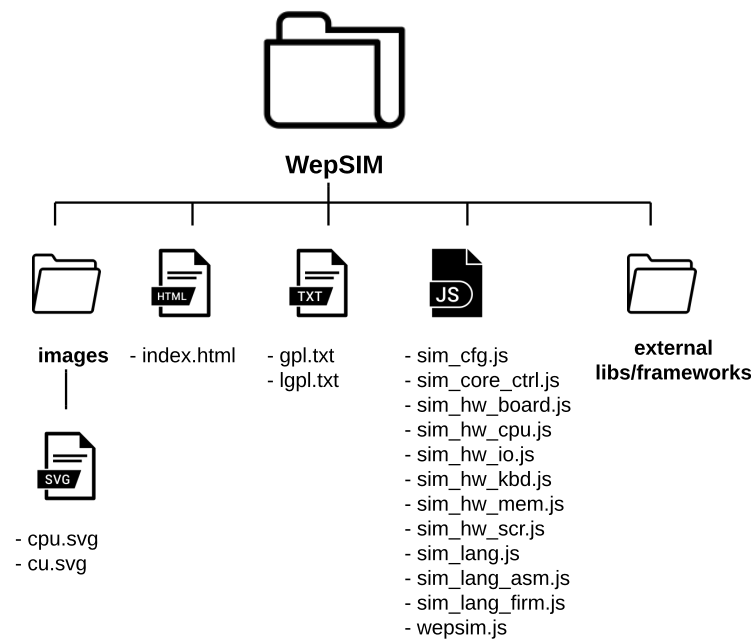


Figura 5-1: Estructura de ficheros.

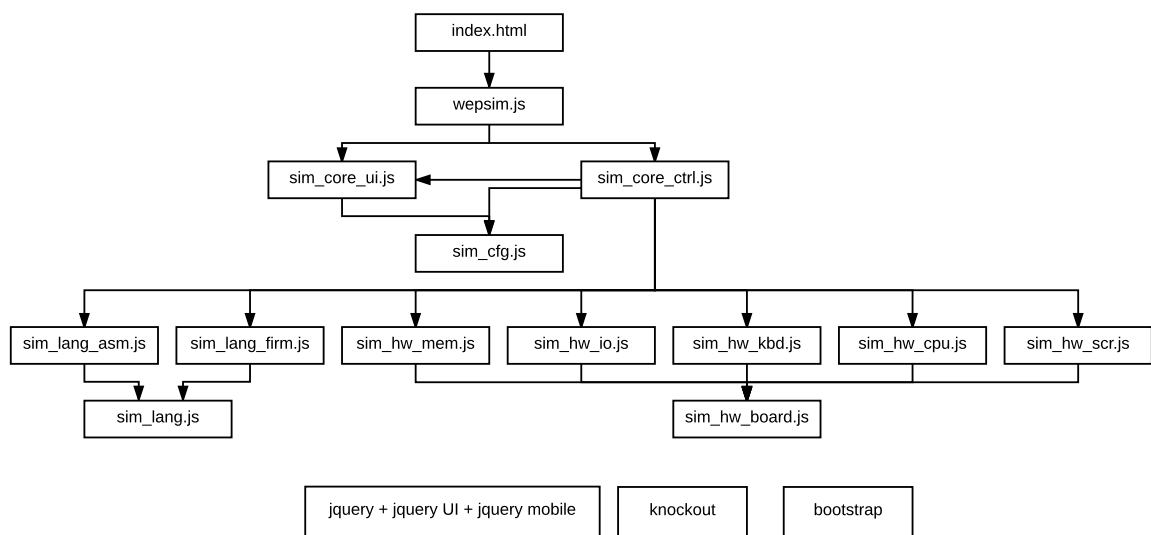


Figura 5-2: Dependencias entre ficheros.

- **gpl.txt y lgpl.txt:** estos ficheros contienen la especificación de las licencias del software.
- **sim_cfg.js** este fichero contiene las estructuras de datos de la configuración de la herramienta.
- **sim_core_ctrl.js:** este fichero contiene la implementación del motor de simulación de la herramienta.
- **sim_core_ui.js:** este fichero contiene la implementación del motor de la interfaz de usuario de la herramienta.
- **sim_hw_cpu.js:** este fichero contiene la definición del modelo hardware de la cpu del simulador.
- **sim_hw_io.js:** este fichero contiene la definición del modelo hardware del módulo de generación de interrupciones del simulador.
- **sim_hw_kbd.js:** este fichero contiene la definición del modelo hardware del módulo del teclado del simulador.
- **sim_hw_scr.js:** este fichero contiene la definición del modelo hardware de la memoria principal del simulador.
- **sim_hw_mem.js:** este fichero contiene la definición del modelo hardware de la pantalla del simulador.
- **sim_lang.js:** este fichero contiene la implementación de las funciones principales del parser de ficheros de la herramienta.
- **sim_lang_asm.js:** este fichero contiene la implementación del compilador de código ensamblador de la herramienta.
- **sim_lang_firm.js:** este fichero contiene la implementación del compilador de firmware de la herramienta.
- **images/cpu.svg:** este fichero contiene la definición de la imagen vectorial de la cpu de la herramienta.
- **images/cpu.svg:** este fichero contiene la definición de la imagen vectorial de la unidad de control de la herramienta.

- **external folder:** este directorio contiene las bibliotecas y frameworks que necesita el simulador para su correcto funcionamiento, como son JQuery, JQueryUI, JQuery Mobile, Knockout y BootStrap.

En [16] se presenta el manual completo de usuario de la herramienta, que incluye la especificación del modelo hardware implementado y la explicación de uso del simulador, indicando algunos ejemplos docentes para aprender el uso de esta herramienta.

Capítulo 6

Verificación, validación y evaluación

Este capítulo detalla la verificación, validación y evaluación del proyecto. En primer lugar, presentamos la verificación y validación del simulador (Sección 6.1, *Verificación y validación*), y detallamos una serie de pruebas que nos permitieron verificar que habíamos cumplido todos los requisitos establecidos en el Capítulo 3 (*Análisis*). Después de esto, mostramos la validación de los resultados de las simulaciones, demostrando que el simulador realiza simulaciones precisas y realistas.

Para la realización de las simulaciones, hemos tomado la definición de un juego de instrucciones base proporcionado por el coordinador de la asignatura Estructura de Computadores, pudiendo así validar que el funcionamiento del simulador es correcto al obtener los resultados esperados en cada una de las instrucciones y códigos simulados en la herramienta.

6.1 Verificación y validación

El principal objetivo de esta sección es verificar que todos los requisitos detallados en el Capítulo 3 (*Análisis*) han sido cumplidos. Además, validamos los resultados obtenidos con WepSIM, comparándolos con los resultados teóricos esperados de la definición del juego de instrucciones de la asignatura Estructura de Computadores y los ejercicios especificados en el libro [11].

En la ingeniería de software, la verificación y validación son los procesos de comprobar que un sistema de software cumple con las especificaciones y que cumple con su propósito. Como se explica en el Capítulo 3 (*Análisis*), el cliente fija inicialmente los requisitos deseados para el producto final (requisitos del usuario). A partir de ahí, los analistas especifican los requisitos de

software (requisitos funcionales y no funcionales). Para verificar que se cumplen los requisitos del proyecto, se necesitan procesos de verificación y validación (ver Figura 6-1).

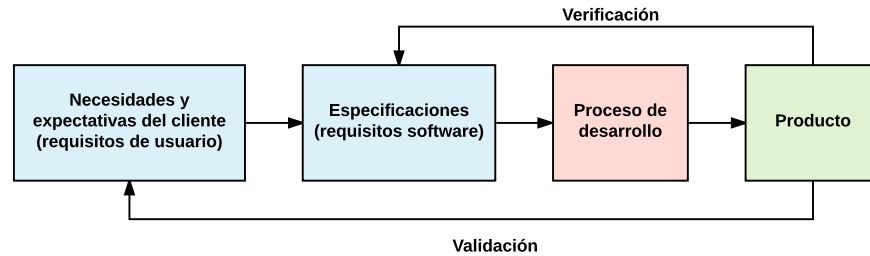


Figura 6-1: *Verificación y validación software.*

Verificación Software Es el proceso de evaluación de los productos de trabajo (no el producto final real) de una fase de desarrollo para determinar si cumplen con los requisitos especificados para esa fase (los requisitos de software). *Validación software* es el proceso de evaluación del producto final al final del proceso de desarrollo para determinar si satisface los requisitos especificados por el usuario al inicio del proyecto [17].

6.1.1 Pruebas de verificación

Con el fin de realizar las pruebas de verificación, hemos seguido un proceso dinámico durante la fase de desarrollo del software. Con estas pruebas queríamos responder a la pregunta: "¿Estamos construyendo el producto correctamente?". La tabla 6.1 proporciona la plantilla utilizada para los test de verificación. Tenga en cuenta que el formato del atributo ID es VET-XX, donde XX indica el número de test de verificación.

ID	Test ID.
Nombre	Nombre del test.
Requisitos	Requisitos de software cumplidos con este test.
Descripción	Descripción del test.
Precondiciones	Las condiciones que siempre deben ser verdad antes de realizar el test.
Procedimiento	Una secuencia fija, paso a paso, de las actividades realizadas por el test.
Postcondiciones	Las condiciones que siempre deben ser verdaderas justo después de realizar el test.
Evaluación	<i>OK</i> or <i>Error</i> .

Tabla 6.1: *Plantilla de pruebas de verificación.*

A continuación, especificamos las pruebas de verificación.

ID	VET-01.
Nombre	Plataforma.
Requisitos	SR-NF-PL01, SR-NF-PL02
Descripción	Verificar que el software puede ser utilizado en los navegadores especificados en los requisitos.
Precondiciones	1. Utilizar una máquina con sistema operativo Ubuntu 16.04 / Windows 10 / MacOS 10.2.5 . 2. Tener instalados los navegadores Microsoft Edge 30+, Mozilla Firefox 45+, Google Chrome 50+ o Safari 10+.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La simulación ha sido completada satisfactoriamente sin ningún error de ejecución debido a la plataforma .
Evaluación	OK

Tabla 6.2: Test de verificación VET-01.

ID	VET-02.
Nombre	Ejecución local.
Requisitos	SR-NF-PL03, SR-NF-PL04
Descripción	Verificar que el software es ejecutado en local sin la necesidad de conexión a internet.
Precondiciones	1. No tener acceso a Internet. 2. Tener el código fuente de la herramienta descargado en local.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La simulación ha sido completada satisfactoriamente sin ningún error de ejecución debido a la conexión de red.
Evaluación	OK

Tabla 6.3: Test de verificación VET-02.

ID	VET-03.
Nombre	Lenguaje de programación HTML5.
Requisitos	SR-NF-PL05
Descripción	Verificar que el software ha sido desarrollando utilizando HTML5.
Precondiciones	1. Tener el código fuente de la herramienta descargado en local.
Procedimiento	1. Comprobar el código de los archivos fuente del simulador.
Postcondiciones	El simulador ha sido desarrollado mediante el lenguaje de programación HTML5.
Evaluación	OK

Tabla 6.4: *Test de verificación VET-03.*

ID	VET-04.
Nombre	Interfaz de usuario.
Requisitos	SR-NF-UI01
Descripción	Verificar que la interfaz del simulador es compatible tanto con pc como con plataformas móviles.
Precondiciones	1. Disponer de un pc y de un dispositivo móvil. 2. Los dispositivos deben tener instalados los navegadores web especificados previamente.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La interfaz del simulador es compatible tanto como pc como con dispositivos móviles, visualizándose correctamente en cualquier pantalla de la herramienta.
Evaluación	OK

Tabla 6.5: *Test de verificación VET-04.*

ID	VET-05.
Nombre	Tiempo medio por ciclo de reloj.
Requisitos	SR-NF-P01
Descripción	Verificar que el tiempo medio por ciclo de reloj del simulador no supera 0,1 segundos.
Precondiciones	1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal utilizando un código ensamblador que utiliza cada una de las partes de la arquitectura del simulador. 4. Ejecutar simulación midiendo el tiempo de ejecución.
Postcondiciones	El tiempo de ejecución medio por ciclo de reloj excede 0,1 segundos.
Evaluación	OK

Tabla 6.6: *Test de verificación VET-05.*

ID	VET-06.
Nombre	Arquitectura del simulador.
Requisitos	SR-F-F01, SR-F-02, SR-F-03
Descripción	Verificar que el software simula una arquitectura de 32 bits con unidad de control microprogramable (con secuenciamiento implícito, saltos a nivel de microdirección y micros saltos condicionales).
Precondiciones	1. Tener el código fuente de la herramienta descargado en local. 2. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente.
Procedimiento	1. Comprobar el código de los archivos fuente del simulador que contienen el modelo hardware a simular (sim_hw_*.js). 2. Generar firmware que define instrucciones de acceso a memoria, operaciones con el banco de registros, uso de los buses de la arquitectura del simulador, saltos a nivel de microdirección y micros saltos condicionales. 3. Generar memoria principal utilizando un código ensamblador que utiliza cada una de las partes de la arquitectura del simulador y las instrucciones de prueba definidas. 4. Verificar que los resultados obtenidos de la ejecución son los resultados teóricos esperados.
Postcondiciones	El modelo hardware ha sido definido correctamente, obteniendo los resultados esperados tras la simulación.
Evaluación	OK

Tabla 6.7: *Test de verificación VET-06.*

ID	VET-07.
Nombre	Definición de juego de instrucciones.
Requisitos	SR-F-F04, SR-F-F05, SR-F-F09, SR-F-F10, SR-F-F11, SR-F-F15
Descripción	Verificar que el software permite la definición del juego de instrucciones en el formato utilizado en la asignatura Estructura de Computadores.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Disponer de un juego de instrucciones base en el formato utilizado en la asignatura Estructura de Computadores. 3. Disponer de un código ensamblador que utiliza el juego de instrucciones utilizado en la asignatura Estructura de Computadores.
Procedimiento	<ol style="list-style-type: none"> 1. Generar firmware mediante el juego de instrucciones proporcionado. 2. Generar memoria principal utilizando el código ensamblador proporcionado. 3. Ejecutar simulación. 4. Verificar que los resultados obtenidos de la ejecución son los resultados teóricos esperados.
Postcondiciones	La memoria de control y la memoria principal han sido generadas correctamente, realizándose la simulación de forma satisfactoria.
Evaluación	OK

Tabla 6.8: *Test de verificación VET-07.*

ID	VET-08.
Nombre	Importación y exportación.
Requisitos	SR-F-F07, SR-F-F08, SR-F-F13, SR-F-F14
Descripción	Verificar que el software permite tanto la importación como la exportación de/a fichero del juego de instrucciones y del código ensamblador.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Disponer de un juego de instrucciones base en el formato utilizado en la asignatura Estructura de Computadores en un fichero. 3. Disponer de un código ensamblador que utiliza el juego de instrucciones utilizado en la asignatura Estructura de Computadores en un fichero.
Procedimiento	<ol style="list-style-type: none"> 1. Importar juego de instrucciones mediante el fichero proporcionado. 2. Generar firmware. 3. Exportar juego de instrucciones. 4. Importar código ensamblador mediante el fichero proporcionado. 5. Generar memoria principal. 6. Exportar código ensamblador. 7. Verificar que la memoria de control y la memoria principal han sido generadas correctamente. 8. Verificar que los ficheros exportados han sido generados correctamente.
Postcondiciones	La memoria de control, la memoria principal y los dos ficheros exportados han sido generados correctamente.
Evaluación	OK

Tabla 6.9: *Test de verificación VET-08.*

ID	VET-09.
Nombre	Edición de texto.
Requisitos	SR-F-F06, SR-F-F12
Descripción	Verificar que el software permite la edición del juego de instrucciones y del código ensamblador.
Precondiciones	1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente.
Procedimiento	<ol style="list-style-type: none"> 1. Definir un juego de instrucciones mediante la herramienta. 2. Generar firmware. 3. Definir código ensamblador mediante la herramienta. 4. Generar memoria principal.
Postcondiciones	La memoria de control, la memoria principal han sido generadas correctamente, pudiéndose editar desde la herramienta.
Evaluación	OK

Tabla 6.10: *Test de verificación VET-09.*

ID	VET-10.
Nombre	Tipos de simulaciones.
Requisitos	SR-F-F16, SR-F-F17, SR-F-F18, SR-F-F22
Descripción	Verificar que el software permite los tres tipos de simulación: ciclo a ciclo de reloj, instrucción a instrucción y simulación completa.
Precondiciones	1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	1. Realizar la simulación del código cargado microinstrucción a microinstrucción. 2. Reiniciar la simulación. 3. Realizar la simulación del código cargado instrucción a instrucción. 4. Reiniciar la simulación. 5. Realizar la simulación completa del código cargado.
Postcondiciones	Las simulaciones han sido realizadas correctamente, obteniéndose los resultados teóricos esperados en ellas.
Evaluación	OK

Tabla 6.11: *Test de verificación VET-10.*

ID	VET-11.
Nombre	Velocidad de simulación.
Requisitos	SR-F-F19
Descripción	Verificar que el software permite la configuración de la velocidad de simulación.
Precondiciones	1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	1. Realizar la simulación completa del código cargado. 2. Reiniciar la simulación. 3. Modificar velocidad de simulación. 4. Realizar la simulación completa del código cargado.
Postcondiciones	Las simulaciones han sido realizadas correctamente, obteniéndose diferentes tiempos en función de la velocidad de simulación establecida.
Evaluación	OK

Tabla 6.12: *Test de verificación VET-11.*

ID	VET-12.
Nombre	Información de las simulaciones.
Requisitos	SR-F-F20, SR-F-21, SR-F-22
Descripción	Verificar que el software muestra los esquemas de la CPU y la Unidad de control, modificando el color de las señales activas y buses actualizados y muestra la información del estado de los registros.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	1. Realizar la simulación del código cargado ciclo a ciclo de reloj.
Postcondiciones	Las simulaciones han sido realizadas correctamente, cambiándose el color de las señales activas y buses de datos actualizados en cada ciclo de reloj y mostrándose la información de los registros.
Evaluación	OK

Tabla 6.13: *Test de verificación VET-12.*

La matriz de trazabilidad de las pruebas de verificación (Tabla 6.14) Determina que todos los requisitos de software se han verificado durante la fase de desarrollo del proyecto.

Requirements	VET-01	VET-02	VET-03	VET-04	VET-05	VET-06	VET-07	VET-08	VET-09	VET-10	VET-11	VET-12
SR-F-F01						✓						
SR-F-F02						✓						
SR-F-F03						✓						
SR-F-F04							✓					
SR-F-F05							✓					
SR-F-F06									✓			
SR-F-F07								✓				
SR-F-F08								✓				
SR-F-F09							✓					
SR-F-F10							✓					
SR-F-F11							✓					
SR-F-F12									✓			
SR-F-F13								✓				
SR-F-F14								✓				
SR-F-F15							✓					
SR-F-F16										✓		
SR-F-F17										✓		
SR-F-F18										✓		
SR-F-F19											✓	
SR-F-F20												✓
SR-F-F21												✓
SR-F-F22										✓		✓
SR-NF-PL01	✓											
SR-NF-PL02	✓											
SR-NF-PL03		✓										
SR-NF-PL04		✓										
SR-NF-PL05			✓									
SR-NF-UI01				✓								
SR-NF-P01					✓							

Tabla 6.14: Matriz de trazabilidad de pruebas de verificación.

6.1.2 Pruebas de validación

Para realizar las pruebas de validación, hemos comprobado el software final, comparándolo con las necesidades del usuario especificadas en el Capítulo 3 *Análisis*). Con estas pruebas queremos responder a la pregunta: "¿Hemos construido el producto adecuado? ". La tabla 6.15 proporciona la plantilla utilizada para los test de validación. Tenga en cuenta que el formato del atributo ID es VAT-XX, donde XX indica el número de test de validación.

ID	Test ID.
Nombre	Nombre del test.
Requisitos	Requisitos del usuario cumplidos con esta prueba.
Test de verificación	Pruebas de verificación que nos ayudan a validar esta prueba.
Descripción	Descripción de la prueba.
Precondiciones	Las condiciones que siempre deben ser verdad antes de realizar la prueba.
Procedimiento	Una secuencia fija, paso a paso, de las actividades realizadas por la prueba.
Postcondiciones	Las condiciones que siempre deben ser verdaderas justo después de realizar la prueba.
Evaluación	OK or Error.

Tabla 6.15: *Plantilla para test de validación.*

A continuación, especificamos las pruebas de validación.

ID	VAT-01.
Nombre	Juego de instrucciones.
Requisitos	UR-C02, UR-C03
Descripción	Validar que el software permite la definición del juego de instrucciones en el formato utilizado en la asignatura Estructura de Computadores mediante la carga de un fichero y la edición desde la herramienta, generando la memoria de control asociada y permitiendo la exportación del juego de instrucciones a un fichero.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Disponer de un juego de instrucciones base en el formato utilizado en la asignatura Estructura de Computadores.
Procedimiento	<ol style="list-style-type: none"> 1. Importar juego de instrucciones desde fichero. 2. Editar el juego de instrucciones desde la herramienta. 3. Generar memoria de control utilizando el juego de instrucciones cargado. 4. Exportar el juego de instrucciones cargado a fichero.
Postcondiciones	La memoria de control ha sido generada correctamente, permitiéndose la carga desde fichero, la edición desde la herramienta y la exportación del juego de instrucciones a fichero.
Evaluación	OK

Tabla 6.16: *Test de validación VAT-01.*

ID	VAT-02.
Nombre	Juego de instrucciones.
Requisitos	UR-C04, UR-C05
Descripción	Validar que el software permite la definición del código ensamblador en el formato utilizado en la asignatura Estructura de Computadores mediante la carga de un fichero y la edición desde la herramienta, generando la memoria principal asociada y permitiendo la exportación del código ensamblador a un fichero.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Disponer de un juego de instrucciones base en el formato utilizado en la asignatura Estructura de Computadores cargado en el simulador. 3. Disponer de un código ensamblador base en el formato utilizado en la asignatura Estructura de Computadores.
Procedimiento	<ol style="list-style-type: none"> 1. Importar código ensamblador desde fichero. 2. Editar el código ensamblador desde la herramienta. 3. Generar memoria principal utilizando el código ensamblador cargado. 4. Exportar el código ensamblador cargado a fichero.
Postcondiciones	La memoria principal ha sido generada correctamente, permitiéndose la carga desde fichero, la edición desde la herramienta y la exportación del código ensamblador a fichero.
Evaluación	OK

Tabla 6.17: *Test de validación VAT-02.*

ID	VAT-03.
Nombre	Simulación del modelo hardware propuesto.
Requisitos	UR-C01
Descripción	Validar que el software permite la simulación del modelo hardware propuesto.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Disponer de un juego de instrucciones base en el formato utilizado en la asignatura Estructura de Computadores cargado en el simulador. 3. Disponer de un código ensamblador base en el formato utilizado en la asignatura Estructura de Computadores cargado en el simulador.
Procedimiento	<ol style="list-style-type: none"> 1. Comprobar el código de los archivos fuente del simulador que contienen el modelo hardware a simular (sim_hw_*.js). 2. Ejecutar la simulación del código ensamblador.
Postcondiciones	El modelo hardware definido se corresponde con el modelo propuesto, siendo el resultado de la simulación el resultado teórico esperado.
Evaluación	OK

Tabla 6.18: *Test de validación VAT-03.*

ID	VAT-04.
Nombre	Tipos de simulaciones.
Requisitos	UR-C06
Descripción	Verificar que el software permite los tres tipos de simulación: ciclo a ciclo de reloj, instrucción a instrucción y simulación completa.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	<ol style="list-style-type: none"> 1. Realizar la simulación del código cargado microinstrucción a microinstrucción. 2. Reiniciar la simulación. 3. Realizar la simulación del código cargado instrucción a instrucción. 4. Reiniciar la simulación. 5. Realizar la simulación completa del código cargado.
Postcondiciones	Las simulaciones han sido realizadas correctamente, obteniéndose los resultados teóricos esperados en ellas.
Evaluación	OK

Tabla 6.19: *Test de validación VAT-04.*

ID	VAT-05.
Nombre	Velocidad de simulación.
Requisitos	UR-C07
Descripción	Verificar que el software permite la configuración de la velocidad de simulación.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	<ol style="list-style-type: none"> 1. Realizar la simulación completa del código cargado. 2. Reiniciar la simulación. 3. Modificar velocidad de simulación. 4. Realizar la simulación completa del código cargado.
Postcondiciones	Las simulaciones han sido realizadas correctamente, obteniéndose diferentes tiempos en función de la velocidad de simulación establecida.
Evaluación	OK

Tabla 6.20: *Test de validación VAT-05.*

ID	VAT-06.
Nombre	Información de las simulaciones.
Requisitos	UR-C08, UR-C09
Descripción	Verificar que el software muestra los esquemas de la CPU y la Unidad de control, modificando el color de las señales activas y buses actualizados y muestra la información del estado de los registros.
Precondiciones	<ol style="list-style-type: none"> 1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente. 2. Tener la memoria de control y la memoria principal de la herramienta generadas.
Procedimiento	1. Realizar la simulación del código cargado ciclo a ciclo de reloj.
Postcondiciones	Las simulaciones han sido realizadas correctamente, cambiándose el color de las señales activas y buses de datos actualizados en cada ciclo de reloj y mostrándose la información de los registros.
Evaluación	OK

Tabla 6.21: *Test de validación VAT-06.*

ID	VAT-07.
Nombre	Plataforma.
Requisitos	UR-R01, UR-R02
Descripción	Verificar que el software puede ser utilizado en los navegadores especificados en los requisitos.
Precondiciones	1. Utilizar una máquina con sistema operativo Ubuntu 16.04 / Windows 10 / MacOS 10.2.5 . 2. Tener instalados los navegadores Microsoft Edge 30+, Mozilla Firefox 45+, Google Chrome 50+ o Safari 10+.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La simulación ha sido completada satisfactoriamente sin ningún error de ejecución debido a la plataforma .
Evaluación	OK

Tabla 6.22: *Test de validación VAT-07.*

ID	VAT-08.
Nombre	Interfaz de usuario.
Requisitos	UR-R03
Descripción	Verificar que la interfaz del simulador es compatible tanto con pc como con plataformas móviles.
Precondiciones	1. Disponer de un pc y de un dispositivo móvil. 2. Los dispositivos deben tener instalados los navegadores web especificados previamente.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La interfaz del simulador es compatible tanto como pc como con dispositivos móviles, visualizándose correctamente en cualquier pantalla de la herramienta.
Evaluación	OK

Tabla 6.23: *Test de validación VAT-08.*

ID	VAT-09.
Nombre	Tiempo medio por ciclo de reloj.
Requisitos	UR-R04
Descripción	Verificar que el tiempo medio por ciclo de reloj del simulador no supera 0,1 segundos.
Precondiciones	1. Disponer de un dispositivo con una versión de navegador web igual o superior a los indicados anteriormente.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal utilizando un código ensamblador que utiliza cada una de las partes de la arquitectura del simulador. 4. Ejecutar simulación midiendo el tiempo de ejecución.
Postcondiciones	El tiempo de ejecución medio por ciclo de reloj excede 0,1 segundos.
Evaluación	OK

Tabla 6.24: *Test de validación VAT-09.*

ID	VAT-10.
Nombre	Ejecución local.
Requisitos	UR-R05, UR-R06
Descripción	Verificar que el software es ejecutado en local sin la necesidad de conexión a internet.
Precondiciones	1. No tener acceso a Internet. 2. Tener el código fuente de la herramienta descargado en local.
Procedimiento	1. Abrir el software en cualquiera de los navegadores web especificados. 2. Generar firmware. 3. Generar memoria principal. 4. Ejecutar simulación.
Postcondiciones	La simulación ha sido completada satisfactoriamente sin ningún error de ejecución debido a la conexión de red.
Evaluación	OK

Tabla 6.25: *Test de validación VAT-10.*

ID	VAT-11.
Nombre	Tecnologías de desarrollo.
Requisitos	UR-R07
Descripción	Verificar que el software ha sido desarrollando utilizando HTML5.
Precondiciones	1. Tener el código fuente de la herramienta descargado en local.
Procedimiento	1. Comprobar el código de los archivos fuente del simulador.
Postcondiciones	El simulador ha sido desarrollado mediante el lenguaje de programación HTML5.
Evaluación	OK

Tabla 6.26: *Test de validación VAT-11.*

La matriz de trazabilidad de las pruebas de validación (Tabla 6.27) Determina que todos los requisitos de usuario han sido validados con el producto final.

Requirements	VAT-01	VAT-02	VAT-03	VAT-04	VAT-05	VAT-06	VAT-07	VAT-08	VAT-09	VAT-10	VAT-11
UR-C01			✓								
UR-C02	✓										
UR-C03	✓										
UR-C04		✓									
UR-C05		✓									
UR-C06				✓							
UR-C07					✓						
UR-C08						✓					
UR-C09						✓					
UR-R01							✓				
UR-R02							✓				
UR-R03								✓			
UR-R04									✓		
UR-R05										✓	
UR-R06										✓	
UR-R07											✓

Tabla 6.27: Matriz de trazabilidad de pruebas de validación.

Capítulo 7

Planificación y presupuesto

Este capítulo presenta una planificación detallada del proyecto (Sección 7.1, *Planificación*). Luego, explicamos los costes del proyecto (Sección 7.2, *Presupuesto*). Al final del capítulo, explicamos el entorno socio-económico del proyecto (Section 7.3, *Entorno Socio-Económico*).

7.1 Planificación

Esta sección incluye la planificación completa del proyecto. En primer lugar, describimos la metodología de desarrollo de software utilizada. Después, detallamos la duración de cada fase del proyecto, indicando todos los tiempos en un gráfico de Gantt.

7.1.1 Justificación de la Metodología

Debido a sus características, hemos dividido nuestro proyecto en tres iteraciones:

- **Modelo hardware:** la primera iteración ha consistido en lograr el modelado del hardware que compone el procesador WepSIM. El objetivo de esta fase, ha sido lograr el modelado y funcionamiento básico de cada componente hardware por separado.
- **Modelo software:** esta fase ha consistido en lograr el modelo software utilizado en la asignatura Estructura de computadores para la definición del juego de instrucciones y el lenguaje ensamblador. El objetivo de esta fase, ha sido lograr interpretar el juego de instrucciones definido y código ensamblador definidos por el usuario, generando la memoria de control y memoria principal en el orden correspondiente.

- **Kernel del simulador:** esta fase ha consistido en lograr unir el modelo hardware y modelo software, realizando el kernel del simulador. El objetivo de esta fase, ha sido diseñar e implementar el kernel de simulador encargado de unir el modelo hardware y el modelo software, generando la simulación y la vista del simulador.

Era necesario contar con una metodología iterativa para desarrollar cada una de las fases de manera independiente para unir las todas en la última etapa y obtener el producto final. Para ello, hemos analizado tres metodologías de desarrollo de software diferentes: Prototipado de software [18], el Modelo en Cascada [19] y el Modelo en Espiral [20]. El Prototipado de Software no encaja muy bien en nuestro proyecto, puesto que requiere la construcción de un prototipo de software en poco tiempo. El Modelo en Cascada es un proceso de diseño secuencial, utilizado en procesos de desarrollo de software en el cual el progreso es visto como un flujo constante hacia abajo (como una cascada) a través de diferentes fases. El problema con esta metodología es que no permite iteraciones dentro del desarrollo del software. Finalmente, el Modelo en Espiral permitió dividir el proyecto en diferentes iteraciones. Este modelo, combina las fortalezas de los otros dos modelos (simplicidad y flexibilidad), utilizando además un proceso iterativo. Aunque este modelo es más lento que los dos anteriores, nos permitió aplicar diferentes iteraciones por lo que decidimos aplicarlo a todo el proceso.

7.1.2 Ciclo de Vida

El proceso de desarrollo del ciclo de vida del proyecto ha seguido el Modelo de ciclo de vida en Espiral [20]. La Figura 7-1 muestra el Modelo en Espiral usando un esquema.

El modelo en Espiral tiene cuatro fases, que se repiten durante las diferentes iteraciones del modelo. Estas fases son:

- **Planificación** ('Determine objectives' en Figura 7-1): Se reúnen los requisitos de los usuarios, se realiza un estudio de factibilidad del sistema y se determinan los objetivos de iteración.
- **Análisis** ('Identify and resolve risks' en Figura 7-1): Se realiza un análisis completo de los requisitos y se identifican los posibles riesgos. Esta fase termina con un diseño básico.
- **Desarrollo y Pruebas:** Se lleva a cabo la implementación del código. Se realizan los casos de prueba y los resultados de la prueba.

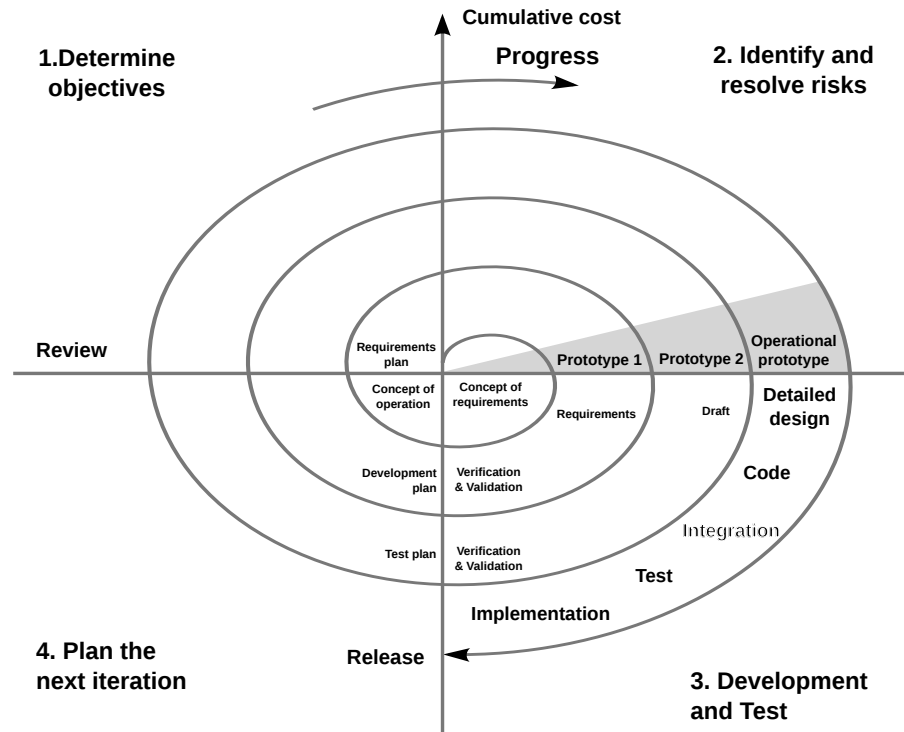


Figura 7-1: *Modelo en Espiral* (Boehm, 2000).

- **Evaluación** ('Plan the next iteration' en Figura 7-1): Los clientes evalúan el software y proporcionan sus comentarios. En este caso, el estudiante intenta obtener la aprobación del supervisor. Esta es la *tarea crítica* del ciclo de vida, ya que solo podemos pasar a la siguiente iteración del Modelo de ciclo de vida Espiral si esta tarea es aprobada.

Cada fase comienza con un objetivo de diseño y termina con el cliente (el supervisor) revisando el progreso hasta ahora. Como se explicó anteriormente, hemos dividido el desarrollo de software en tres iteraciones: modelo hardware, modelo software, y kernel del simulador. En la última iteración, el software completo debe someterse a pruebas exhaustivas para validar el simulador.

7.1.3 Tiempo Estimado

El gráfico de Gantt (Figura 7-2) muestra todas las tareas realizadas durante el desarrollo del proyecto. El proyecto comenzó el 2 de noviembre de 2015 y finalizó el 30 de diciembre de 2016, lo que supone un total de casi 13 meses de trabajo (305 días laborables). Durante este tiempo, he trabajado de lunes a viernes, durante cuatro horas al día.

El diagrama de Gantt muestra todas las tareas realizadas en cada iteración del modelo de ciclo de vida espiral. Recuerde que las tres iteraciones fueron: Modelo hardware, Modelo software y kernel del simulador. Además de las tareas (fases) mencionadas anteriormente (Planificación, Análisis, Desarrollo y Pruebas, y Evaluación), hemos incluido la tarea de Documentación al final de cada iteración. La tarea de documentación ha consistido principalmente en la redacción de este trabajo fin de grado.

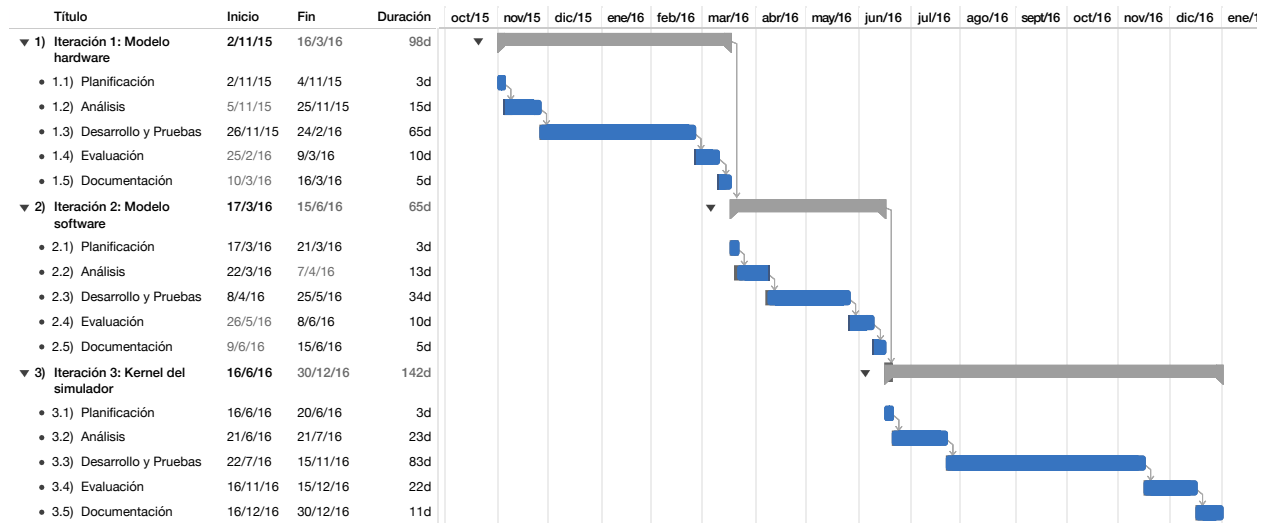


Figura 7-2: Diagrama Gantt.

7.2 Presupuesto

Esta sección detalla el presupuesto general del proyecto. Por un lado, presentamos los costes del proyecto y, por otro lado, damos a conocer la oferta presentada al cliente.

7.2.1 Coste del Proyecto

La tabla 7.1 resume las principales características del proyecto, incluido el presupuesto total.

A continuación se desglosa el presupuesto total del proyecto.

<i>Información del Proyecto</i>	
Título	WepSIM: Simulador de procesador elemental con unidad de control microprogramada
Autor	Javier Prieto Cepeda
Departamento	Departamento de Informática
Fecha de inicio	2 de Noviembre de 2015
Fecha de finalización	30 de Diciembre de 2016
Duración	13 meses
Ratio de costes indirectos	20 %
Presupuesto total	51,860.96

Tabla 7.1: *Información del Proyecto.*

Costes Directos

En esta parte se presentan los costos directos del proyecto. La Tabla 7.2 muestra los costes directos causados por los costes de personal, sobre la base de la planificación presentada en la sección anterior. El tutor y el estudiante han desempeñado los siguientes roles:

- **Tutor:** Jefe de Proyecto.
- **Estudiante:** Analista, Desarrollador, Tester.

Categoría	Coste por hora (€)	Horas	Total (€)
Jefe de Proyecto	60	65	3,900
Analista	35	269	9,415
Desarrollador	35	523	18,305
Tester	25	428	10,700
Total			42,320.00

Tabla 7.2: *Costes de recursos humanos.*

La tabla 7.3 muestra los costes directos causados por la adquisición y uso del equipo. El coste amortizado, C , es calculado utilizando la siguiente formula:

$$C = \frac{d \cdot c \cdot u}{D} \quad (7.1)$$

Donde:

- **C:** Coste amortizado. Es equivalente al valor amortizado.
- **d:** Tiempo que el equipamiento ha sido utilizado.
- **c:** Coste del equipamiento.
- **u:** Dedicación al proyecto. Porcentaje del tiempo que el equipamiento ha sido utilizado.
- **D:** Periodo de amortización del equipamiento.

Concepto	Coste, c (€)	Dedicación, u (%)	Dedicación, d (meses)	Amortización, D (meses)	Coste amortizado, C (€)
Desktop PC	799.99	100	13	36	288.89
Laptop	529.99	25	13	36	47.85
ARCOS Mirlo	1979.99	50	7	60	115.50
Impresora	79.99	5	4	60	0.27
Total					452.50

Tabla 7.3: Costes de equipamiento.

Además, el equipo presentado en la Tabla 7.3 es detallado a continuación:

- **Desktop PC:** All in One - Asus Z220ICUK, 21.5", i5-6400T, 8GB, 1TB)
- **Laptop:** Toshiba L50D-C-19D, A10-8700P, 8GB RAM and 1TB.
- **ARCOS Mirlo:** Servidor utilizado por el grupo de investigación ARCOS. 32GB RAM and eight i7 processors of 2.67GHz each.
- **Impresora:** Brother DCP-1510.

Otros costes directos se muestran en la Tabla 7.4. Estos costes consisten en material de oficina, un tóner para la impresora y el abono transporte mensual. El material de oficina incluye: lápices, bolígrafos, cuadernos, papel, tipex y marcadores.

Concepto	Coste (€)
Material de oficina	112.98
Toner (x1)	71.99
Abono transporte (x13)	260
Total	444.97

Tabla 7.4: *Otros costes directos.*

Resumen de Costes

La Tabla 7.5 muestra el resumen completo de los costes del proyecto. Los costes indirectos (20 % de los costes directos) consisten en las facturas de electricidad y agua, teléfono, acceso a internet, etc.

<i>Resumen de costes</i>	
Recursos humanos	42,320.00
Equipamiento	452.50
Otros costes directos	444.97
Costes indirectos	8,643.49
Presupuesto total	51,860.96

Tabla 7.5: *Resumen de costes.*

El presupuesto total de este proyecto asciende a **51,860.96 € (cincuenta y uno mil ochocientos sesenta euros y noventa y seis céntimos)**.

7.2.2 Oferta de Proyecto Propuesta

La Tabla 7.6 una propuesta de oferta detallada. Esta oferta incluye los riesgos estimados (20 %), los beneficios esperados (15 %), y el Impuesto de Valor Agregado (Impuesto Sobre el

Valor Añadido (IVA)), que corresponde al 21 % [21]. Después de aplicar todos estos conceptos, la cantidad final para este proyecto en caso de venta a un cliente de terceros es **86,597.43 € (Ochenta y seis mil quinientos noventa y siete euros y cuarenta y tres céntimos)**.

<i>Oferta propuesta</i>				
Concepto		Incremento (%)	Valor Parcial (€)	Coste agregado (€)
Costes del proyecto		-	51,860.96	51,860.96
Riesgos		20	10,372.19	62,233.15
Beneficios		15	9,334.97	71,568.13
IVA		21	15,029.31	86,597.43
Total				86,597.43

Tabla 7.6: *Oferta propuesta.*

7.3 Entorno Socio-Económico

Como se ha comentado en capítulos anteriores, WepSIM puede apoyar la enseñanza en Estructura y Arquitectura de Computadores. Esto significa que los profesores de estas asignaturas pueden utilizar esta herramienta como refuerzo en las explicaciones del temario para mostrar de forma visual y práctica a los alumnos el comportamiento de un procesador elemental. Además, sirve para que los alumnos puedan desarrollar las prácticas de estas asignaturas sin la necesidad de utilizar diferentes herramientas, evitando la pérdida de tiempo que esto supone y permitiendo a los alumnos dedicar el mayor tiempo posible a la realización de los ejercicios.

Debido al diseño modular realizado de este simulador, WepSIM permite la especificación tanto de diferentes juegos de instrucciones como de diferentes modelos hardware. Esto supone un gran avance en el área docente, puesto que permite el uso de una misma herramienta para la enseñanza y simulación de diferentes dispositivos. Anteriormente se invertía mucho dinero para que los alumnos pudieran realizar prácticas con diferentes tipos de procesadores, ya que se debían de adquirir los diferentes componentes hardware necesarios para la realización de las prácticas. Con WepSIM este concepto cambia, puesto que modelando el comportamiento

del computador deseado, puede ser utilizado para diferentes tipos de prácticas, evitando costes de adquisición y mantenimiento y permitiendo una mayor variedad de opciones a la hora de diseñar las prácticas.

Gracias al uso de este simulador en el ámbito docente, se puede mostrar el impacto energético que supone una programación eficiente. Esto es debido a que WepSIM es un simulador de microprogramación y programación en ensamblador, pudiendo hacer visible el coste a bajo nivel de unas malas prácticas de programación. El objetivo, es concienciar a los alumnos de la importancia que tiene la optimización y eficiencia en el uso de los recursos de un computador, puesto que uno de los retos de la comunidad científica es diseñar sistemas cada vez más eficientes energéticamente.

Capítulo 8

Conclusiones y trabajos futuros

En este capítulo presentamos las conclusiones del trabajo, revisamos los objetivos establecidos al principio de este documento, e incluimos algunas conclusiones personales. Además, discutiremos las principales contribuciones de nuestro trabajo, indicando también las publicaciones resultantes de este trabajo. Finalmente, discutimos el trabajo futuro.

8.1 Conclusiones

En este trabajo se ha descrito el diseño de WepSIM, un simulador de un procesador elemental con unidad de control microprogramada. Este trabajo presenta un nuevo simulador que resulta intuitivo, portable y extensible, sirviendo como complemento docente para la docencia en Estructura de Computadores. Este simulador, permite definir diferentes juegos de instrucciones y ejecutar y depurar código fuente que use el conjunto de instrucciones definido. También permite definir el comportamiento del procesador mediante microprogramación.

WepSIM, permite a los estudiante entender el funcionamiento de un procesador elemental de una forma sencilla, pudiendo ser usado desde un dispositivo móvil o un ordenador con un navegador Web moderno, sin la necesidad de ser instalado. De esta forma, los estudiantes pueden interactuar con el simulador aprendiendo y comprendiendo el funcionamiento del procesador elemental WepSIM, incluyendo los mecanismos de interacción con el software de sistema e integrando en una misma herramienta tanto la microprogramación como la programación en ensamblador.

El objetivo principal de este proyecto era desarrollar un simulador, que a diferencia de los existentes, pudiera simular de forma completa el comportamiento de un procesador elemental

permitiendo comprobar el estado de los componentes en cada ciclo de reloj, de manera que ayudáse a los alumnos a comprender y asimilar de forma sencilla y visual el funcionamiento de un procesador. También hemos cumplido con todos los demás objetivos presentados en la introducción del documento:

- Simular la ejecución de el juego de instrucciones especificado en un computador denominado WepSIM desde el punto de vista de la micropogramación y la programación en ensamblador.
- Permitir la especificación de diferentes juegos de instrucciones.
- Permitir unificar la microprogramación y la programación en ensamblador.
- Permitir al usuario la visualización en cada ciclo de reloj del estado y comportamiento del computador simulado.

A nivel personal, este trabajo me ha ayudado a empezar en el mundo de la investigación científica. He logrado aplicar una gran cantidad de los conocimientos adquiridos a lo largo del grado. Además, he aprendido importantes técnicas de modelado de hardware, compilación y simulación, las cuáles tienen una gran utilidad y complejidad y me han servido para profundizar aún más en los conocimientos adquiridos en el grado. Por todo ello, es muy satisfactorio ver el resultado final obtenido, puesto que he logrado superar todos los problemas que han surgido a lo largo del proyecto.

8.1.1 Contribuciones

El proyecto llevado a cabo durante este Trabajo Fin de Grado encaja con muchas de las asignaturas estudiadas en el Grado en Ingeniería Informática de la Universidad Carlos III de Madrid, destacando los siguientes temas en particular:

- **Tecnología de Computadores** (asignatura obligatoria, Primer curso) en donde se introducen los componentes hardware y la lógica binaria.
- **Estructura de Computadores** (asignatura obligatoria, Segundo curso) en donde se introducen las bases de la estructura y funcionamiento de un computador.
- **Teoría de Autómatas y Lenguajes Formales** (asignatura obligatoria, Segundo curso) en donde se introducen las bases acerca de los lenguajes y gramáticas formales.

- **Sistemas Operativos** (asignatura obligatoria, Segundo curso) en donde se introducen las bases del funcionamiento del sistema operativo.
- **Arquitectura de Computadores** (asignatura obligatoria, Tercer curso) en donde se introducen las bases de la arquitectura de un computador.
- **Diseño de Sistemas Operativos** (asignatura obligatoria, Tercer curso) en donde se introducen las bases del diseño de los distintos módulos de un sistema operativo.
- **Dirección de proyectos de desarrollo de software** (asignatura obligatoria, Tercer curso) en donde se introducen las bases para la dirección y gestión de un proyecto de desarrollo de software.

El proyecto llevado a cabo durante esta tesis de licenciatura encaja con muchas de las materias estudiadas en el Grado de Informática e Ingeniería de la Universidad Carlos III de Madrid, destacando los siguientes temas en particular:

8.1.2 Publicaciones

Este Trabajo Fin de Grado ha hecho una importante contribución al campo de la docencia en Estructura y Arquitectura de Computadores. Las principales contribuciones de este trabajo se han publicado en dos artículos:

- **A. Calderón, F. García-Carballeira, and J. Prieto**, “WepSIM: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador”, *Enseñanza y aprendizaje de ingeniería de computadores*, vol. 6, 35-53, 2016. [15]
- **J. Prieto, A. Calderón, F. García-Carballeira, and S. Alonso-Monsalve**, “WepSIM: simulador integrado de microprogramación y programación en ensamblador”, *Jornadas sarteco 2016*. [22]

Además, en el momento de la entrega de este documento también otro artículo enviado a la espera de su aceptación.

8.2 Trabajos Futuros

Actualmente, hay varias líneas de trabajos futuros en las cuáles estamos trabajando:

- Añadir un módulo de corrección automática de prácticas a la herramienta.
- Añadir revisión semántica del código, pudiendo identificar y notificar los errores de programación al usuario.
- Estudiar el ensamblador de MIPS/ARM generado con gcc/clang de forma que pueda ser usado directamente en WepSIM.
- Introducir más elementos hardware, como por ejemplo una caché.

Bibliografía

- [1] "P8080e simulator." https://www.datsi.fi.upm.es/docencia/Estructura/U_Control/#HERRAMIENTAS. Accessed: 2017-04-26.
- [2] R.-F. Yen and Y. Kim, "Development and implementation of an educational simulator software package for a specific microprogramming architecture," *IEEE Transactions on Education*, no. 1, pp. 1–11, 1986.
- [3] A. S. Tanenbaum, *Structured Computer Organization 2nd*. ACM, 1984.
- [4] J. R. Larus, *Spim s20: A mips r2000 simulator*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin, 1990.
- [5] I. Aguilar Juárez and J. R. Heredia Alonso, "Simuladores y laboratorios virtuales para ingeniería en computación," 2013.
- [6] K. Vollmar and P. Sanderson, "Mars: an education-oriented mips assembly language simulator," in *ACM SIGCSE Bulletin*, vol. 38, pp. 239–243, ACM, 2006.
- [7] S. R. Vegdahl, "Mipsipilot: A compiler-oriented mips simulator," *Journal of Computing Sciences in Colleges*, vol. 24, no. 2, pp. 32–39, 2008.
- [8] M. I. Garcia, S. Rodríguez, A. Pérez, and A. García, "p88110: A graphical simulator for computer architecture and organization courses," *IEEE Transactions on Education*, vol. 52, no. 2, pp. 248–256, 2009.
- [9] I. Branovic, R. Giorgi, and E. Martinelli, "Webmips: a new web-based mips simulation environment for computer architecture education," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, p. 19, ACM, 2004.
- [10] J. C. Garrido, "Arquitectura y diseño de sistemas web modernos," *InforMAS, Revista de Ingeniería Informática del CIIRM*, no. 1, 2004.
- [11] J. C. PEREZ, F. G. CARBALLEIRA, J. D. G. Sánchez, and D. E. Singh, *Problemas resueltos de estructura de computadores*. Ediciones Paraninfo, SA, 2015.
- [12] Institute for Electrical and Electronics Engineers, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE*, pp. 830–1998, 1998.
- [13] BOE, 19 de enero de 2008, "Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999, de 12 de diciembre, de protección de datos de carácter personal," *Boletín Oficial del Estado*, 17:4103-4136.

- [14] "The GNU Lesser General Public License." <http://www.gnu.org/licenses/licenses.html>, Last visited, June 2016.
- [15] A. C. Mateos, F. G. Carballeira, and J. P. Cepeda, "Wepsim: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador," *Enseñanza y aprendizaje de ingeniería de computadores: Revista de Experiencias Docentes en Ingeniería de Computadores*, no. 6, pp. 35–53, 2016.
- [16] ARCOS Research Group , "WepSIM manual user." <https://wepsim.github.io/wepsim-manual.pdf>, Last visited, June 2017.
- [17] Software Testing Fundamentals, "Verification vs Validation." <http://softwaretestingfundamentals.com/verification-vs-validation/>, Last visited, Mar. 2016.
- [18] Accelerated Technologies, Inc, "The Human Condition: A Justification for Rapid Prototyping," *Time Compression Technologies*, vol. 3 no. 3, p. 1, May 1998.
- [19] Benington, Herbert D., "Production of Large Computer Programs," *IEEE Annals of the History of Computing (IEEE Educational Activities Department)*, p. 350–361, Oct. 1983.
- [20] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer* 21, 5, pp. 61–72, 1988.
- [21] BOE, 6 de agosto de 2012, "Resolución de 2 de agosto de 2012, de la Dirección General de Tributos, sobre el tipo impositivo aplicable a determinadas entregas de bienes y prestaciones de servicios en el Impuesto sobre el Valor Añadido," *Boletín Oficial del Estado*, 187:56055-56060.
- [22] J. Prieto-Cepeda, F. Garcia-Carballeira, A. Calderon, and S. Alonso-Monsalve, "WepSIM: simulador integrado de microprogramación y programación en ensamblador," *XXVII Jornadas de Paralelismo (JP2016)*, September 2016.