Trabajo de Fin de Grado

Simulador didáctico de una Arquitectura de planificación estática.

Didactic simulator of a static scheduling architecture.

Melissa Díaz Arteaga

La Laguna, 14 de septiembre de 2018

D. **Iván Castilla Rodríguez**, con N.I.F. 78.565.451-G profesor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Adrián Abreu González**, con N.I.F. 54.111.250-R profesional externo, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"Simulador didáctico de una Arquitectura de planificación estática."

ha sido realizada bajo su dirección por D. **Melissa Díaz Arteaga**,

con N.I.F. 54.113.532-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de septiembre de 2018

Agradecimientos

A Iván Castilla, por su paciencia y estar disposición de ayudarme siempre que lo necesitaba.

A Adrián Abreu, por las tardes dedicadas a arrojarme un poco de luz en este proyecto que compartimos.

A mis profesores, por ofrecerme su tiempo y conocimientos durante estos años en la facultad.

A Rafael, por estar siempre dispuesto a darme ideas desde otro punto de vista cuando no era capaz de verlo.

A mi familia, por aguantar mis horas ausentes del mundo y por apoyarme para lograr cualquier cosa que me proponga.

Licencia

© Esta obra está bajo una licencia de Creative Commons
Reconocimiento 4.0 Internacional.

Resumen

La comprensión del funcionamiento de las arquitecturas de los computadores es imprescindible para ser capaz de aprovechar su funcionalidad. Llegar a conocerlas en su totalidad puede resultar complicado si nos limitamos a aprender conceptos teóricos que en ocasiones pueden ser difíciles de entender.

Por esta razón, el objetivo de este proyecto es hacer el aprendizaje de estas máquinas más interactivo y sencillo, concretamente en el apartado de máquinas de planificación estática de instrucciones. Para ello se ha querido crear un simulador que muestre su funcionamiento como herramienta de apoyo en la docencia.

Para su desarrollo se han utilizado las últimas tecnologías del mundo web tales como React, Redux, TypeScript y Webpack aprovechando así muchas de las mejoras que se han ido incorporando al desarrollo web en los últimos años.

Este trabajo de fin de grado se integra en una línea de desarrollo de simuladores didácticos de arquitecturas de computadores incorporando a SIMDE web un módulo de simulación de arquitectura con planificación estática. Además, se ha querido dar coherencia al proyecto desarrollando una página inicial que permita el acceso a los simuladores disponibles dando la posibilidad de añadir nuevos simuladores que se desarrollen en el futuro.

Palabras clave: Arquitectura de computadores, VLIW, Simulador, Interfaz, Aplicación Web, Planificación estática.

Abstract

An understanding of the functionality of the computer architectures is essential to be able to take advantage of its functionality. Getting to know them in full can be difficult if we limit ourselves to learning theoretical concepts that sometimes can be difficult to understand.

For this reason, this project is aimed at making learning computer architecture more interactive and simple, more specifically in the field of static scheduling of instructions. We achieve this objective by creating a simulator to be used as a teaching tool.

For the development of the simulator, we have used the latest technologies in the web such as React, Redux, TypeScript and Webpack taking advantage of many of the improvements that have been incorporated into web development in recent years.

This end of degree work joins a line of development of teaching simulators of architectures of computers incorporating static scheduling of instructions to SIMDE web. In addition, I have enhanced the coherence to the project developing a home page that allows access to all the simulators available, thus also giving the possibility of adding new simulators that are developed in the future.

Keywords: Computer Architecture, VLIW, Simulator, Interface, Web Application, Static scheduling.

Índice general

Capítulo 1 Introducción 1
1.1 Introducción 1
1.2 Motivación para el trabajo 2
Capítulo 2 Antecedentes 3
2.1 Paralelismo a nivel de instrucción 3
2.1.1 Técnicas de emisión múltiple 5
2.1.2 Planificación dinámica 5
2.1.3 Planificación estática 5
2.1.4 Máquinas Very Long Instruction Word 6
2.2 SIMDE 7
2.3 Otros simuladores 9
Capítulo 3 Objetivos y fases 11
3.1 Objetivos 11
3.2 Fases 12
Capítulo 4 Tecnologías 13
4.1 Lenguajes para la lógica de la aplicación 13
4.1.1 TypeScript 14
4.2 Tecnología para la integración modelo vista 15
4.2.1 React 15
4.2.2 Redux 16
4.3 Tecnología para el ensamblaje 17
4.3.1 Webpack 17
4.4 Tecnologías para la documentación 19
4.4.1 Generadores de contenido estático 19
4.4.2 Hexo 19
4.4.3 Gitbook 20

Capítulo 5 Desarrollo del proyecto 21
5.1 Análisis de requisitos 21
5.2 Proyecto base 23
5.3 Interfaz de integración 25
5.4 Migración de la máquina VLIW 27
5.5 Pruebas Unitarias 28
5.6 Interfaz de visualización 29
5.6.1 Módulos reutilizables 29
5.6.2 Módulos nuevos 34
5.6.3 Integración interfaz - core 35
5.7 Migración de documentación 36
Capítulo 6 Conclusiones y líneas futuras 40
6.1 Conclusiones 40
6.2 Líneas futuras 41
Capítulo 7 Summary and Conclusions 43
7.1 Conclusions 43
7.2 Future work lines 43
Capítulo 8 Presupuesto 45
Capítulo 9 Bibliografía 46

Índice de figuras

Figura 2:1Ejemplo ejecución de instrucciones [1] 3
Figura 2:2 Simulador original SIMDE [4] 7
Figura 2:3 Versión actual SIMDE web [5] 9
Figura 4:1 TypeScript como superconjunto de JavaScript 14
Figura 4:2 Funcionamiento del Virtual DOM de React. [12] 16
Figura 4:3 Funcionamiento de Webpack [14] 18
Figura 5:1 Captura interfaz SIMDE [4] 22
Figura 5:2 Vista inicial SIMDE web 25
Figura 5:3 Vista proyecto SIMDE web (1) 26
Figura 5:4 Vista proyecto SIMDE web (2) 26
Figura 5:5 Ejemplo prueba unitaria 28
Figura 5:6 Access Bar interfaz VLIW 29
Figura 5:7 File Bar interfaz VLIW 29
Figura 5:8 Carga código superescalar y VLIW interfaz VLIW 30
Figura 5:9 Visualizar código superescalar y bloques básicos interfaz VLIW 31
Figura 5:10 Configurar máquina VLIW interfaz VLIW 31
Figura 5:11 Cargar memoria - registro interfaz VLIW 32
Figura 5:12 Configurar ejecución por lotes interfaz VLIW 32
Figura 5:13 Visualizador unidades funcionales interfaz VLIW (recortado) 33
Figura 5:14 Visualizador Memoria- Registros interfaz VLIW 33
Figura 5:15 Visualizador instrucciones largas interfaz VLIW 34
Figura 5:16 Visualizador Predicado y NaT (GPR y FPR) interfaz VLIW 35
Figura 5:17 Interfaz VLIW SIMDE web 36

Figura 5:18 Markdown de máquina VLIW 37
Figura 5:19 Atributos máquina VLIW 37
Figura 5:20 Archivo SUMMARY.md 38
Figura 5:21 Documentación generada 39

Índice de tablas

Tabla 1 Presupuesto 45

Capítulo 1
Introducción

1.1 Introducción

La mejora en la eficiencia ha sido un punto clave en la historia del desarrollo de las arquitecturas de computadores. Aunque tecnológicamente se ha avanzado enormemente en el desarrollo de los computadores actuales, los conceptos en los que se basan sus arquitecturas no han cambiado excesivamente a lo largo del tiempo. Por esta razón es necesario comprender estos conceptos básicos en su totalidad para ser capaz de seguir mejorando las arquitecturas ya existentes.

Sin embargo, el aprendizaje del funcionamiento de las arquitecturas no puede centrarse únicamente en conceptos teóricos, sino que es necesario realizar aproximaciones prácticas difíciles de transmitir utilizando una herramienta docente tradicional. Además, el aumento de la complejidad en la tecnología en los últimos años dificulta aún más su comprensión haciendo imprescindible una abstracción por parte del alumnado al no poder disponer de las máquinas originales.

Este trabajo se centra en el uso de simuladores para la enseñanza del concepto de paralelismo a nivel de instrucción, fundamental en el incremento del rendimiento de los computadores. Para este fin, los simuladores juegan un gran papel, permitiendo añadir a los conceptos teóricos un ámbito práctico donde se puedan reforzar dichos conceptos facilitando la enseñanza en el campo de la Arquitectura de Computadores.

1.2 Motivación para el trabajo

Como se ha mencionado anteriormente, el uso de un simulador para la enseñanza en el área de Arquitectura de Computadores resultaba necesario. Esta necesidad fue detectada por el profesorado de la Universidad de La Laguna, y se plasmó en el desarrollo de un simulador con este propósito.

Desde entonces este simulador ha sido utilizado como herramienta fundamental en la asignatura Arquitectura de Computadores que se imparte en la especialización de Ingeniería de Computadores del Grado de Ingeniería Informática. Se trataba de una aplicación de escritorio desarrollada en C++ que con el paso del tiempo ha quedado obsoleta. No por su funcionalidad, ya que los conceptos en los que se basan estas arquitecturas siguen siendo las mismas sino por su usabilidad y accesibilidad.

Por esta razón se han iniciado una serie de trabajos de fin de grado que buscan recuperar este proyecto para continuar con su desarrollo adaptándolo a las tecnologías web que tanto se utilizan hoy en día. Además, se busca poder darle mantenimiento con el fin de que continúe siendo una herramienta útil y sirva como base para futuros proyectos que enriquezcan la plataforma.

Capítulo 2
Antecedentes

2.1 Paralelismo a nivel de instrucción

Existen diferentes formas de mejorar la eficiencia de una máquina. Una de ellas es aprovechar la potencial capacidad de ejecutar varias instrucciones al mismo tiempo. Esta idea se denomina paralelismo a nivel de instrucción.

Desde hace décadas la mayoría de los computadores han explotado el paralelismo a nivel de instrucción implementando una técnica conocida como segmentación la cual consiste en dividir la ejecución de cada instrucción en varias etapas:

Ciclo	1	2	3	4	5	6	7	8	9
Inst *i*	IF	ID	EX	MM	WB				
Inst *i* + 1		IF	ID	EX	MM	WB			
Inst *i* + 2			IF	ID	EX	MM	WB		
Inst *i* + 3				IF	ID	EX	MM	WB	
Inst *i* + 4					IF	ID	EX	MM	WB

Figura 2:1Ejemplo ejecución de instrucciones [1]

- Captación (*Instruction Fetch - IF*). Lectura de instrucción y actualización del registro contador de programa.

- Decodificación (*Instruction Decode - ID*). Decodificación de la instrucción, lectura de registros, extensión de signo de desplazamientos y cálculo de posible dirección de salto.

- Ejecución (*Execution - EX*). Operación de ALU sobre registros y cálculo de dirección efectiva de salto.

- Memoria (*Memoria - MM*). Lectura o escritura en memoria.

- Post-escritura (*Write-back - WB*). Escritura de resultado en el banco de registros.

Esta técnica afecta mínimamente a la latencia de cada instrucción, pero permite incrementar el *throughput*, puesto que permite finalizar (idealmente) una instrucción por ciclo.

Al ejecutar múltiples instrucciones aparecen riesgos estructurales, de datos o de control que deben detectarse y resolverse de forma adecuada:

- Los riesgos estructurales se producen cuando el hardware no puede soportar todas las posibles secuencias de instrucciones. Esto se produce si dos etapas necesitan hacer uso del mismo recurso *hardware*. En general, los riesgos estructurales se pueden evitar en el diseño, pero encarecen el hardware resultante.

- Un riesgo de datos se produce cuando la segmentación modifica el orden de acceso de lectura/escritura de los operandos. Los riesgos de datos pueden ser de tres tipos RAW (*Read After Write*), WAR (*Write After Read*) y WAW (*Write After Write*).

- Un riego de control se produce en una instrucción de bifurcación cuando no dispone todavía del valor sobre el que se debe tomar la decisión de salto. Los riesgos de control pueden resolverse en tiempo de compilación (soluciones estáticas) o en tiempo de ejecución (soluciones dinámicas). [2]

2.1.1 Técnicas de emisión múltiple

Esta técnica se basa en la emisión de más de una instrucción por ciclo y ofrece varios enfoques posibles: el uso de procesadores superescalares planificados estáticamente, procesadores superescalares planificados dinámicamente y procesadores VLIW.

2.1.2 Planificación dinámica

La planificación dinámica se basa en reordenar, durante la ejecución, las instrucciones de forma que se reduzcan las esperas. Aunque esto implica una mayor complejidad en el *hardware*, permite gestionar las dependencias en tiempo de compilación. Sin embargo, el permitir la ejecución fuera de orden puede introducir riesgos WAR y WAW. Existen dos técnicas de planificación dinámica [2]:

- Técnica *scoreboard*: Detiene las instrucciones emitidas hasta que hay recursos suficientes y no existen riesgos de datos.
- Algoritmo de *Tomasulo*: Elimina las dependencias WAR y WAW realizando renombrado de registros.

2.1.3 Planificación estática

La planificación estática consiste en reordenar las instrucciones de forma que se maximicen el uso de los recursos del computador, eliminando “burbujas o esperas innecesarias. En este tipo de planificación las decisiones se delegan en el *software* más concretamente en el compilador. Esta característica permite simplificar la complejidad del *hardware* dejándole el trabajo al compilador y al propio programador.

La clave en este tipo de planificación está en la construcción de los programas y las técnicas que se emplean para realizar la planificación estática:

- Reordenación de código: Busca aprovechar las características de un cauce con adelantamiento de resultados, eliminando así riesgos estructurales.

- Desenrollado de bucles: Consiste en solapar distintas iteraciones del mismo bucle, es decir, hacer una nueva versión del bucle incluyendo las operaciones de varias iteraciones del bucle original.

- *Software pipelining*: Combina en cada iteración del nuevo bucle operaciones de diferentes iteraciones del código original.

- Predicción estática de saltos: Trata de predecir el comportamiento del salto en tiempo de compilación. Para ello deciden considerar el salto como verdadero siempre; seleccionar el destino en base a si el salto se presupone tomado o no tomado; o almacenar información de ejecuciones anteriores.

Estas técnicas, entre otras ayudan a explotar el paralelismo a nivel de instrucción mediante *software*. [4]

2.1.4 Máquinas Very Long Instruction Word

Las máquinas Very Long Instruction Word se caracterizan por tener juegos de instrucciones muy simples en cuanto a número de instrucciones diferentes pero muy grandes en cuanto a tamaño de cada instrucción.

Esto es así porque en cada instrucción se especifica el estado de todas las unidades funcionales del sistema con el objetivo de simplificar el diseño del *hardware* al dejar todo el trabajo en manos del compilador (planificación estática) al contrario de lo que sucede en las máquinas superescalares en las que el *hardware* es quien planifica las instrucciones en tiempo de ejecución (planificación dinámica).

Una de las grandes desventajas de esta arquitectura es que requiere compiladores muy complejos. Además, cualquier mejora que se realice en el *hardware* implica problemas de compatibilidad hacia atrás de los repertorios de instrucciones de las máquinas. [3]

2.2 SIMDE

Hace algunos años, el profesor Iván Castilla desarrolló un simulador didáctico de arquitecturas de computadores como parte de su proyecto de fin de carrera en la Universidad de La Laguna. Su objetivo era arrojar un poco de luz en el entendimiento de estos temas. Este proyecto fue denominado SIMDE. [4]

Esta aplicación se centró en simular el funcionamiento de las máquinas Superescalares y Very Long Instruction Word principales arquitecturas estudiadas en la asignatura arquitectura de computadores. Este proyecto cumple con las características necesarias en un simulador para la docencia, sin embargo, esta herramienta ya se encuentra desfasada debido a la falta de mantenimiento. Tras su desarrollo utilizando C++98 y C++ Builder no se han realizado actualizaciones ni mantenimiento de la misma.

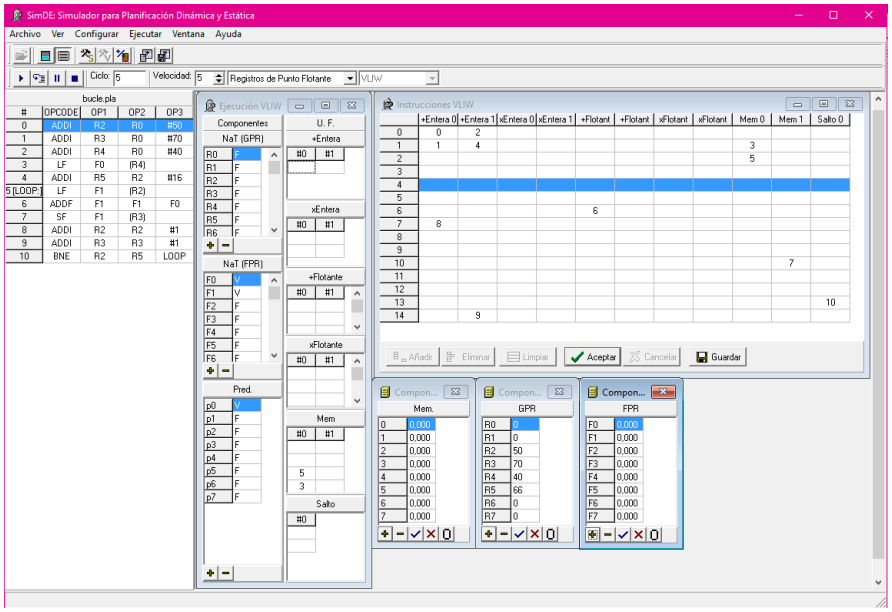


Figura 2:2 Simulador original SIMDE [4]

Durante el análisis de las tecnologías a utilizar en el desarrollo de SIMDE original se tuvo en consideración la utilización de tecnologías web. Sin embargo, esta opción se desechó debido a que el simulador no estaba planteado como una herramienta con arquitectura cliente-servidor, ni tampoco se pretendía una ejecución distribuida. La idea inicial es que se ejecutase en la máquina del alumno y no en un servidor. Descartando además el uso de *applets* de java debido a la lentitud de su ejecución al ser un lenguaje interpretado. [4]

Estos argumentos, aunque eran perfectamente válidos en ese momento hoy en día no están justificados. En la actualidad el rendimiento de JavaScript en los navegadores es realmente alto por lo que se utiliza en la gran mayoría de las aplicaciones web existentes.

En estos trece años que han pasado desde el desarrollo de SIMDE la tecnología ha ido dando pasos de gigantes y si miramos hacia atrás poco tiene que ver el desarrollo web de ahora con el de esa época. La tendencia de migrar todas las aplicaciones a la nube para aumentar su accesibilidad obligó a los navegadores a aumentar su rendimiento y gracias a esto en la actualidad la mayoría de las aplicaciones tienen una versión web accesible desde cualquier parte del mundo.

Por las razones anteriormente expuestas y por la gran utilidad que a día de hoy sigue teniendo SIMDE para los estudiantes de Ingeniería de Computadores se planteó la migración de la aplicación al entorno web adaptándola así a las necesidades actuales.

El año pasado, el alumno del Grado de Ingeniería Informática, Adrián Abreu comenzó como parte de su trabajo de fin de grado este proyecto, encargándose de la migración de la máquina superescalar lo que marcaría las bases para continuar con el desarrollo.

El resultado final fue un simulador de la máquina superescalar similar a la disponible en el SIMDE original. Además, agregó algunas mejoras al funcionamiento de la herramienta:

- Parseador: Mejoró el sistema de errores de SIMDE original mostrando la línea donde se ha producido el error lo que facilita enormemente la construcción de códigos en el simulador.

- Modo de ejecución por lotes: Eliminó el límite de velocidad de la simulación que tenía la versión original.

- Histórico: Agregó la posibilidad de ir hacia atrás en la ejecución de la máquina.

Todas estas mejoras eran resultado de años de uso del simulador en los cuales se pudieron identificar nuevas funcionalidades deseadas por los usuarios de la herramienta. [5]

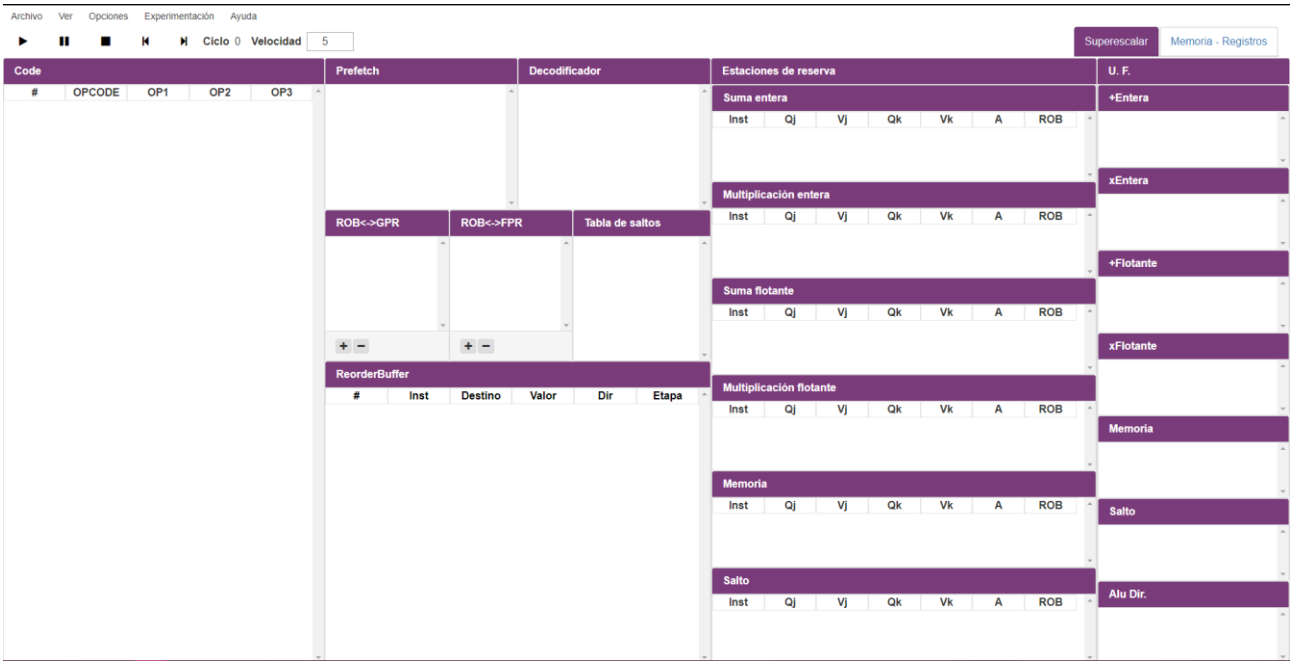


Figura 2:3 Versión actual SIMDE web [5]

2.3 Otros simuladores

Antes de comenzar con la migración de la máquina VLIW a su versión web se realizó una búsqueda de simuladores de esta máquina con el fin de comprobar si existían ya simuladores con estas características.

- VEX: Es un programa gratuito que funciona bajo el sistema operativo Linux. Este programa dispone de un conjunto de herramientas que permiten manejar un entorno de simulación de la arquitectura VLIW. Permite analizar, desarrollar y depurar cualquier algoritmo escrito en C en la arquitectura VLIW.

Ofrece ventajas al alumnado como el poder entender cómo se organizan las instrucciones para aprovechar al máximo los recursos *hardware* de los que dispone el procesador, analizar el rendimiento del procesador VLIW con un programa escrito en alto nivel y ayuda además a entender el comportamiento de la arquitectura superescalar ya que los resultados de este simulador son similares a la ejecución de ese mismo código en un ordenador superescalar.

Sin embargo, VEX presenta una serie de inconvenientes que le dificultan ser considerada como una buena herramienta docente:

- Se usa desde terminal de Linux por lo que necesita comandos textuales que son específicos para cada programa. Son difíciles de recordar y poco intuitivos lo que dificulta su utilización por parte del alumnado.

- Usa ficheros para almacenar los resultados de las simulaciones. Estos archivos son difíciles de interpretar ya que utiliza bastantes líneas para mostrar unas pocas instrucciones VLIW.

- Se deben usar aplicaciones externas como editores externos para escribir los ficheros o cargarlos.

Como podemos ver, la utilidad de VEX se centra en el mismo ámbito que SIMDE web. Sin embargo, VEX carece de una interfaz que muestre la simulación paso a paso, así como de una forma sencilla de ejecutar los programas características más que deseables en un simulador dedicado a la docencia. [6]

- VLIW-DLX: Es un simulador gráfico de los procesadores VLIW dirigido a ser utilizado en cursos de arquitectura de computadores. Está basado en el simulador WinDXL y fue desarrollado en la Universidad de Praga por el Departamento de Ciencia de la Computación e Ingeniería.

Aunque este simulador cuenta con una interfaz con la que interactuar, la versión actual de VLIW-DLX carece de una representación de la máquina superescalar.

- Simple-VLIW es una plataforma de simulación de la máquina VLIW. [8] Se trata de un *software* propietario por lo que no se dispone de mucha información acerca de él. Sin embargo, no se describe como un simulador para el apoyo en la docencia, sino que su finalidad es agilizar la investigación y el desarrollo de arquitecturas VLIW.

Estos dos últimos simuladores son específicos para las máquinas VLIW y, aunque cuentan con interfaz, su usabilidad no se asemeja a la de SIMDE web. El que más se asemeja a nuestro propósito es VEX, un simulador de código abierto para uso docente y que aún tiene que madurar como proyecto para ser capaz de ser útil para este fin.

Capítulo 3
Objetivos y fases

3.1 Objetivos

Inicialmente se planteó un anteproyecto para continuar con la migración de SIMDE que tenía los siguientes objetivos:

1. Estudio de las tecnologías empleadas para la versión actual de SIMDE web.

2. Análisis de requerimientos y diseño de la migración de VLIW.

3. Implementación del *Back-End* del simulador VLIW integrada con SIMDE web.

4. Implementación del *Front-End* del simulador VLIW en SIMDE web.

5. Testeo de la aplicación con pruebas unitarias y de integración.

Tras algunas reuniones con el tutor del proyecto se vio la necesidad de añadir algunos apartados como:

- Creación de una *landing page* para la integración de ambas máquinas como parte del apartado de desarrollo del *Front-End*.

- Recuperación de la documentación de SIMDE sobre VLIW agregándola a la existente de superescalar.

3.2 Fases

El desarrollo del proyecto se dividió finalmente en cinco fases principales:

1. Estudio de las tecnologías empleadas para la versión actual de SIMDE web: El proceso de conocer las tecnologías utilizadas para desarrollar el núcleo de la aplicación (estructuras básicas comunes) y la máquina superescalar así como comprender el código ya desarrollado.

2. Creación de una *landing page* para la integración de ambas máquinas: Como parte del aprendizaje y el desarrollo del *Front-End* se desarrolló una página para dar acceso a ambas máquinas, dando la impresión de una aplicación única de cara al usuario.

3. Análisis de los requerimientos y diseño de la migración de VLIW: Esta fase consistió en estudiar nuevamente la teoría de las máquinas VLIW para comprender su funcionamiento y posteriormente se consultó la documentación disponible sobre SIMDE en la memoria de TFG de mi tutor Iván Castilla y de mi cotutor Adrián Abreu.

4. Implementación del *Back-End* del simulador VLIW integrada con SIMDE web: El proceso completo de reescribir el código de las estructuras necesarias para el funcionamiento de la máquina VLIW en TypeScript.

5. Implementación del *Front-End* del simulador VLIW integrada con SIMDE web: Proceso en el que se implementó la interfaz de visualización de la máquina, así como la interfaz de opciones de configuración de la máquina VLIW.

6. Testeo de la aplicación: Creación de archivos de testeo donde poder incrustar tanto el código en MIPS como el código de instrucciones largas con el fin de comprobar el correcto funcionamiento de la máquina.

   A la vez que se realizaba el testeo de la aplicación se llevó a cabo la recuperación de la documentación de SIMDE original sobre VLIW agregándola a la existente de superescalar.

Capítulo 4
Tecnologías

Una de las partes clave de este proyecto era el estudio y comprensión de las tecnologías empleadas para la migración de la máquina superescalar a su versión web. Por esta razón se realizará un breve análisis sobre las tecnologías escogidas, así como las razones que llevaron a tomar dichas elecciones.

4.1 Lenguajes para la lógica de la aplicación

Cuando hablamos del desarrollo de la funcionalidad de una página web normalmente se relaciona con JavaScript ya que es uno de los lenguajes más utilizados para este fin, sin embargo, no parecía la mejor opción para SIMDE ya que este lenguaje no cuenta con muchas de las funcionalidades que nos da C++ como podría ser la programación orientada a objetos que está muy presente en la antigua versión de SIMDE. [4]

Por esta razón se quiso plantear la posibilidad de utilizar un lenguaje que permitiera este tipo de programación facilitando así la migración.

4.1.1 TypeScript

TypeScript es un lenguaje libre y de código abierto desarrollado por Microsoft que actúa como un superconjunto de JavaScript escrito que se compila en JavaScript, es decir, incorpora los distintos estándares: ECMA5, ECMA6, ECMA7… [9]. Algunas características destacables de TypeScript son:

- Es un lenguaje compilado: a diferencia de JavaScript, TypeScript permite detectar muchos de los errores antes de abordar una ejecución lo que hace más simple el desarrollo y depuración de proyectos.

- Programación orientada a objetos: soporta el manejo de clases, interfaces, herencia y un largo etcétera que hace de TypeScript un lenguaje fuerte y sólido. Finalmente se decidió utilizar esta tecnología por las siguientes razones:

1. TypeScript tiene bastante apoyo por parte de la comunidad y por parte de la propia empresa Microsoft. La documentación es extensa y efectiva

2. TypeScript está alineada en cierta forma con el futuro de JavaScript. Microsoft es uno de los muchos que forman parte del consenso de estándar de ECMASCRIPT.

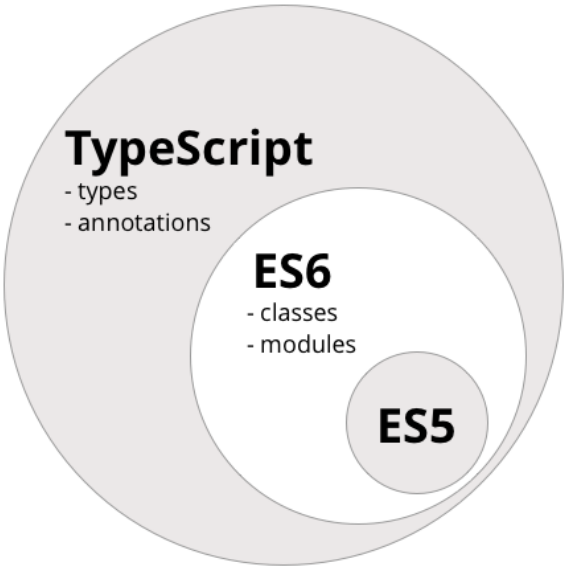


Figura 4:1 TypeScript como superconjunto de JavaScript

3. TypeScript no establece ningún límite en la posibilidad de usar JavaScript, todo código JavaScript es código TypeScript válido [5].

Al introducirme en este proyecto he podido comprender las razones que llevaron a escoger este lenguaje para el proyecto y es que engloba todo lo bueno de JavaScript con la utilidad de la programación orientada a objetos de lenguajes como C++.

4.2 Tecnología para la integración modelo vista

Como parte del desarrollo de la aplicación se quiso introducir una librería que se encargara del proceso de manipulación del DOM con el fin de facilitar el desarrollo de interfaces de cara al usuario.

4.2.1 React

React es una librería JavaScript de código abierto para el desarrollo de interfaces. [10] Fue creada por Facebook con el fin de facilitar la creación de componentes interactivos y reutilizables para interfaces de usuario.

React.js se basa en hacer funciones que toman las actualizaciones de estado y las traduce en una representación virtual de la página resultante. Cada vez que React detecta un cambio de estado en la página vuelve a ejecutar esas funciones que realizan cambios en el DOM para reflejar la nueva página.

Aunque este método pueda parecer lento, React cuenta con un algoritmo muy eficiente para determinar las diferencias entre la página actual y la nueva realizando el menor número de cambios en el DOM. Concretamente utiliza un concepto llamado DOM virtual que hace sub-árboles de nodos sobre la base de cambios de estado con el fin de mantener los datos actualizados. [11]

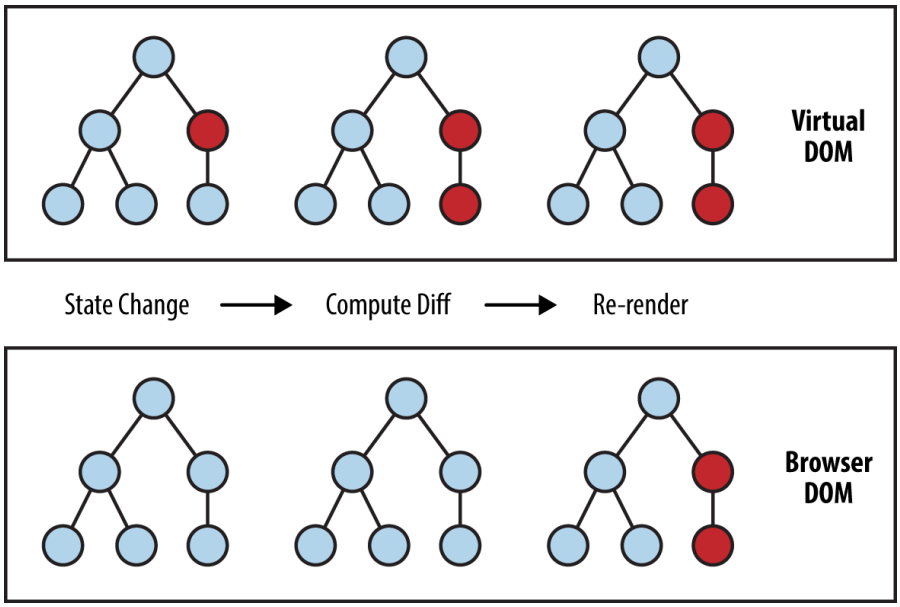


Figura 4:2 Funcionamiento del Virtual DOM de React. [12]

En este proyecto se ha decidido utilizar React ya que tiene una excelente integración con TypeScript utilizando el formato tsx. [5] Además, esta tecnología cuenta con una amplia comunidad lo que facilita la implementación y resolución de problemas a la hora de buscar información.

4.2.2 Redux

Redux es una librería para controlar el estado de una aplicación web de forma sencilla, esta librería está pensada para React.js aunque también se utiliza en Angular.js e incluso en Vanilla JS.

Redux centraliza todos los eventos que pretenden cambiar el estado de la aplicación los cuales, en vez de hacer cambios directamente, envían intenciones de cambio. De esta forma Redux puede gestionarlas emitiendo una por una y evitar que los cambios se solapen. [13] Esta tecnología es muy fácil de usar y muy útil para la gestión de interfaces, razón por la cual fue escogida para el desarrollo de SIMDE web.

4.3 Tecnología para el ensamblaje

Para generar la aplicación final es necesario realizar una serie de pasos intermedios. Para este proyecto concretamente se necesita:

- Compilar el código TypeScript para tener su versión en JavaScript.
- Compilar el código. tsx (TypeScript integrado con React) a .jsx (JavaScript integrado con React.)
- Resolver las dependencias existentes en el proyecto.
- Procesar el código SASS para convertirlo en código CSS. [5]

4.3.1 Webpack

Webpack se define como un *empaquetador de módulos*, pero hace muchas cosas más como:

- Gestión de dependencias.
- Ejecución de tareas.
- Conversión de formatos.
- Servidor de desarrollo.
- Carga y uso de módulos de todo tipo (AMD, CommonJS o ES2015)
- …

Nació a finales de 2012 y en la actualidad es utilizado por miles de proyectos de desarrollo web *Front-End*. Esta herramienta es muy útil para aplicaciones web desarrolladas de forma modular, es decir, separan el código en módulos que luego se utilizan como dependencias en otros módulos ya que Webpack se encarga de colocar un grafo de dependencias a todos los elementos que forman parte del proyecto: código JavaScript, HTML, CSS, plantillas, imágenes, fuentes… [14]

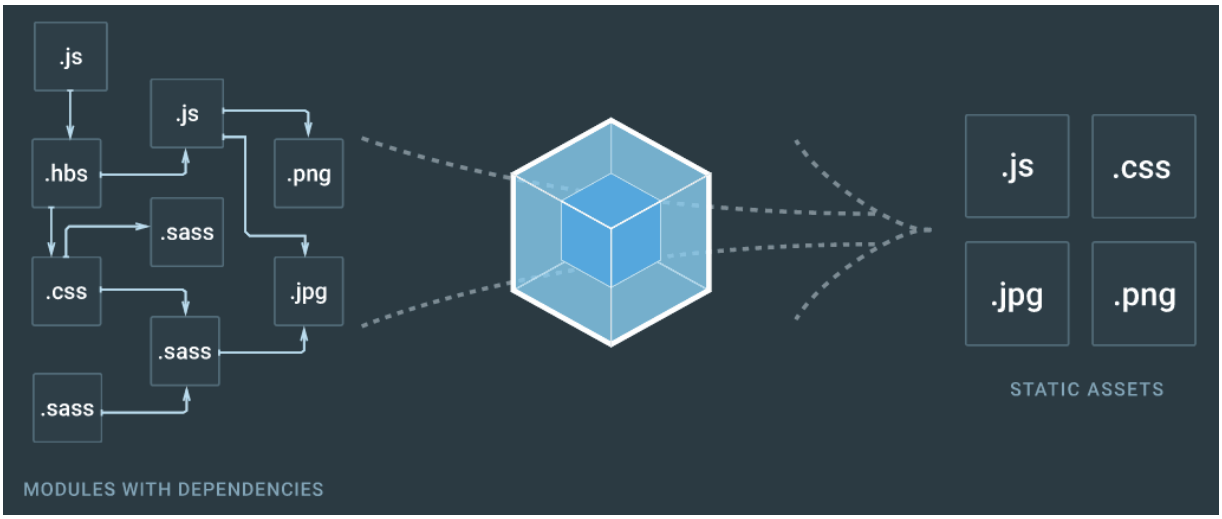


Figura 4:3 Funcionamiento de Webpack [14]

Esta idea es lo que hace de Webpack una herramienta tan potente y una opción perfecta para ser utilizada en este proyecto ya que se trata de una aplicación dividida en módulos tanto en la realización del *Back-End* como en la implementación de la interfaz.

4.4 Tecnologías para la documentación

Después de realizar la migración de una aplicación a su versión web parece necesario migrar también la documentación disponible a web. Aunque existen una infinidad de generadores de documentación, SIMDE contaba con una documentación bastante extensa por lo que al migrar la documentación inicial de SIMDE web se optó por utilizar generadores estáticos de documentación.

4.4.1 Generadores de contenido estático

Los generadores de contenido estático se encargan básicamente de generar ficheros HTL y CSS a partir de una serie de ficheros fuentes escritos en un formato de entrada como podría ser Markdown.

Este tipo de generadores tienen grandes ventajas para los desarrolladores como el hecho de que no es necesario mantener una base de datos o múltiples ficheros HTML. Además, estos generadores tienen algunas otras ventajas como la velocidad de carga de la página debido a que todo el sitio web consta de archivos HTML estáticos que están en el servidor y que cargan al instante al recibir una petición. [15]

Estas características entre otras hacen de los generadores de contenido estático una opción muy atractiva para desarrolladores que quieren realizar sitios webs simples, que carguen rápido, que sean más seguros y más fáciles de mantener como podría ser un blog o una página de presentación de un curriculum vitae. [16]

4.4.2 Hexo

Hexo un *framework* rápido, simple y potente que tras escribir publicaciones en Markdown (u otros lenguajes) genera archivos estáticos en segundos. [17]

Este generador de contenido estático fue escogido inicialmente para el proyecto de SIMDE web porque al estar basado en JavaScript parecía la mejor opción para crear homogeneidad en el desarrollo del proyecto. [5] Sin embargo, poco después se estudiaron tecnologías para la generación de documentación y esta se migró finalmente a Gitbook.

4.4.3 Gitbook

Gitbook es una herramienta para crear documentación de proyectos y libros técnicos utilizando Markdown y Git/GitHub. Permite incluir ejemplos y ejercicios interactivos para posteriormente publicarlos en GitHub u otro hosting web. [18] Teniendo además la opción de exportar el resultado a e-book o pdf.

Está programada en Node.JS y actualmente existe además una aplicación de escritorio que facilita la creación de libros y su publicación. También ofrece una plataforma web que nos permite publicar nuestras creaciones para que otras personas puedan verlas de forma gratuita. [19]

Debido a la magnitud del proyecto, una de las características más deseables de Gitbook es la posibilidad de crear archivos independientes para cada módulo nuevo pudiéndolo conectar con el resto de la documentación por unas sencillas reglas. De esta forma no es necesario modificar la documentación existente para añadir nueva documentación.

Además, se trata de un sistema desarrollado por terceros que no es necesario estar manteniendo y es muy fácil su integración en el repositorio. Por las razones anteriormente expuestas se decidió finalmente adaptar la generación de documentación a esta herramienta.

Capítulo 5
Desarrollo del proyecto

5.1 Análisis de requisitos

Para comenzar a desarrollar el proyecto, era necesario comprobar las características con las que contaba el simulador que queríamos migrar. Se identificaron las siguientes opciones:

- Cargar código VLIW: Permite cargar desde fichero un código VLIW.

- Construir código VLIW: Permite crear un nuevo código VLIW en la aplicación.

- Configurar maquina VLIW: Permite configurar el número de unidades funcionales con las que cuenta la máquina.

- Cargar/Salvar contenido Memoria/Registro: Permite cargar/salvar desde fichero el contenido de la memoria y los registros.

Para la visualización del funcionamiento de la máquina VLIW existen varios módulos:

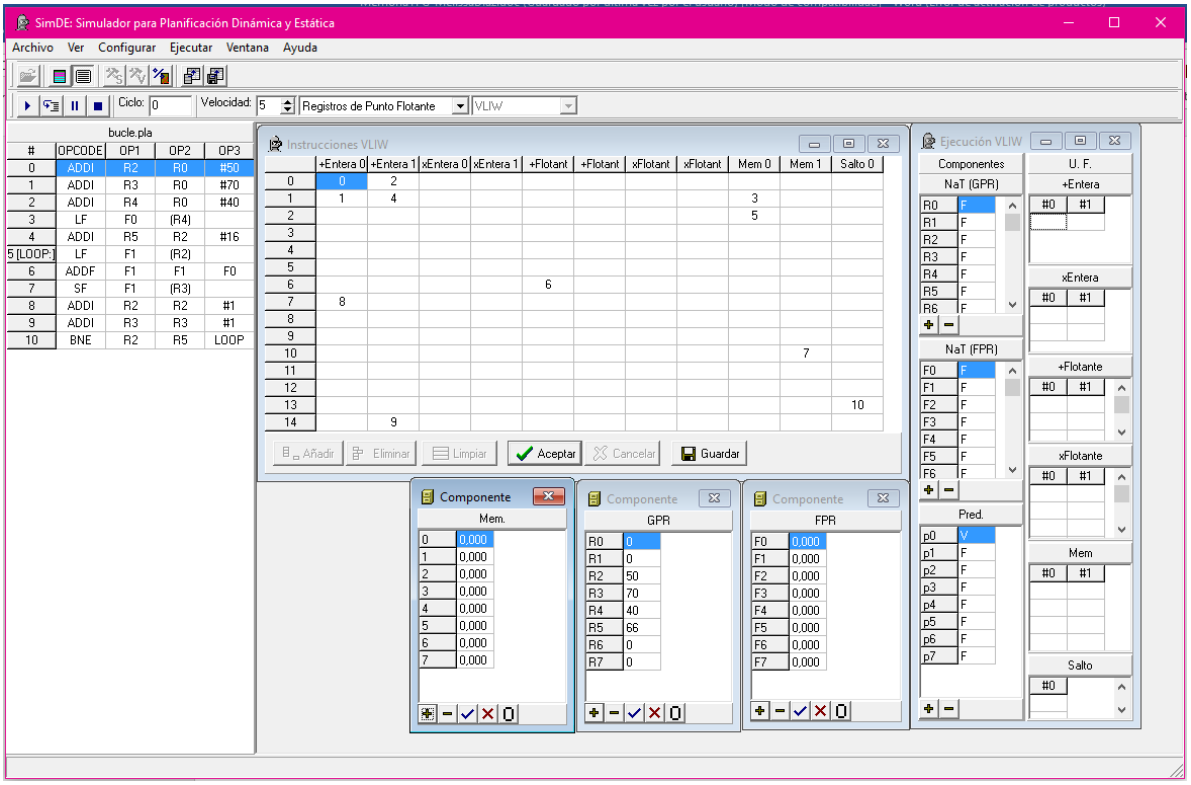


Figura 5:1 Captura interfaz SIMDE [4]

- Planificador: Módulo que permite visualizar y crear las instrucciones largas que va a ejecutar la máquina VLIW.

- Predicado: Módulo que permite visualizar el estado del predicado durante la ejecución.

- NaTGPR: Módulo que permite visualizar el estado del bit NaTGPR durante la ejecución.

- NaTFPR: Modulo Módulo que permite visualizar el estado del bit NaTFPR durante la ejecución.

De esta forma se identificaron los módulos que se debían implementar en la nueva versión de SIMDE.

5.2 Proyecto base

Para llevar a cabo este trabajo se tomó como proyecto base el repositorio de etsii/SIMDE donde se encontraba alojado el trabajo ya realizado en la aplicación SIMDE web. Debía comenzar analizando lo que se había hecho ya que ambas máquinas tenían estructuras comunes reutilizables. Los componentes fundamentales del proyecto original que se han reutilizado en este trabajo son:

- Code.ts: Gestiona e identifica las instrucciones del código superescalar. En la simulación de la máquina VLIW se utiliza el código superescalar para posteriormente poder realizarse la planificación y asociar las instrucciones planificadas a su correspondiente instrucción superescalar. Para la implementación de la máquina VLIW se creó una clase específica con este mismo fin.

- FunctionalUnit.ts: Se encarga de gestionar las unidades funcionales con las que cuenta la máquina. Ambas máquinas cuentan con distintas unidades funcionales que se encargan de realizar los cálculos. Cada una de ellas se gestiona de forma independiente pudiendo existir varias unidades funcionales de un mismo tipo.

- Instruction.ts: Desglosa las instrucciones superescalares identificando los operandos de la misma. También identifica otras características de las instrucciones como el id permitiendo además diferenciar los diferentes bloques básicos existentes en el código superescalar. Para la máquina VLIW era necesario implementar una clase específica que gestionara las instrucciones largas las cuales están compuestas por varias instrucciones superescalartes.

- Machine.ts: Gestiona todos los elementos necesarios para el buen funcionamiento de la máquina superescalar. Aunque este módulo no se utiliza directamente, en la máquina VLIW el funcionamiento es distinto, aunque la clase VLIW creada con este fin hereda algunas de las características disponibles en Machine.

- MachineStatus.ts: Se encarga de identificar los diferentes estados de la máquina.

- Memory.ts: Gestiona la memoria de la máquina, en ambas máquinas se utiliza el mismo módulo de memoria permitiendo ver el estado de esta durante toda la ejecución.

- Opcodes.ts: Se encarga de identificar los *opcodes* en las instrucciones del código superescalar.

- Parser.ts: Identifica el tipo de instrucción superescalar que se necesita ejecutar gestionando posteriormente sus operandos. Para el funcionamiento de la máquina VLIW fue necesario crear un parser especifico con el mismo fin.

- Register.ts: Gestiona el funcionamiento de los diferentes registros de la máquina manteniendo el estado de los mismos durante la ejecución.

- Status.ts: Gestiona el estado de las instrucciones identificando las instrucciones que deben ejecutarse en el siguiente ciclo.

Después de analizar y comprender el código ya desarrollado era el momento de instalar las dependencias del mismo y así empezar a hacer pruebas para familiarizarme con las tecnologías utilizadas. Para ello bastó con situarme en el directorio raíz del proyecto y ejecutar el comando:

```
$ npm install
```

Una vez instaladas las dependencias ejecuté el comando:

```
$ npm start
```

Este comando arranca un servidor embebido en Webpack que permite visualizar el proyecto en un navegador refrescándose de forma automática cada vez que guardamos un cambio.

Esta funcionalidad es muy útil cuando se está desarrollando la interfaz de la aplicación, pero no es muy recomendable durante el desarrollo *Back-End* debido a que se están realizando cambios continuamente que no necesariamente se quieren visualizar.

5.3 Interfaz de integración

Inicialmente, con el fin de familiarizarme con las tecnologías utilizadas en el proyecto decidí crear una interfaz para la integración de ambas máquinas. Parecía lógico pensar que para ser capaces de aumentar el número de máquinas simuladas en SIMDE web y que a su vez siguiese pareciendo una única aplicación era necesario crear una página inicial de la aplicación que redirigiese a cada una de las máquinas implementadas.

Por esta razón desarrollé lo que se denomina una *landing page* o página de aterrizaje muy simple que permitiera añadir en un futuro nuevas implementaciones añadiendo únicamente un módulo nuevo para cada una de las máquinas que se quieran agregar a SIMDE web.

La interfaz cuenta con una página inicial donde se encuentra una pequeña presentación de la aplicación junto con un botón *Leer más* que nos redirige al apartado *Proyecto*. Justo debajo podemos ver dos *"cartas"* que corresponden con las máquinas de las que este simulador dispone, cada una de ellas con una pequeña descripción, un botón *Ir* que redirige al propio simulador de cada máquina y un botón *Leer más* que nos redirige a la documentación de la aplicación.

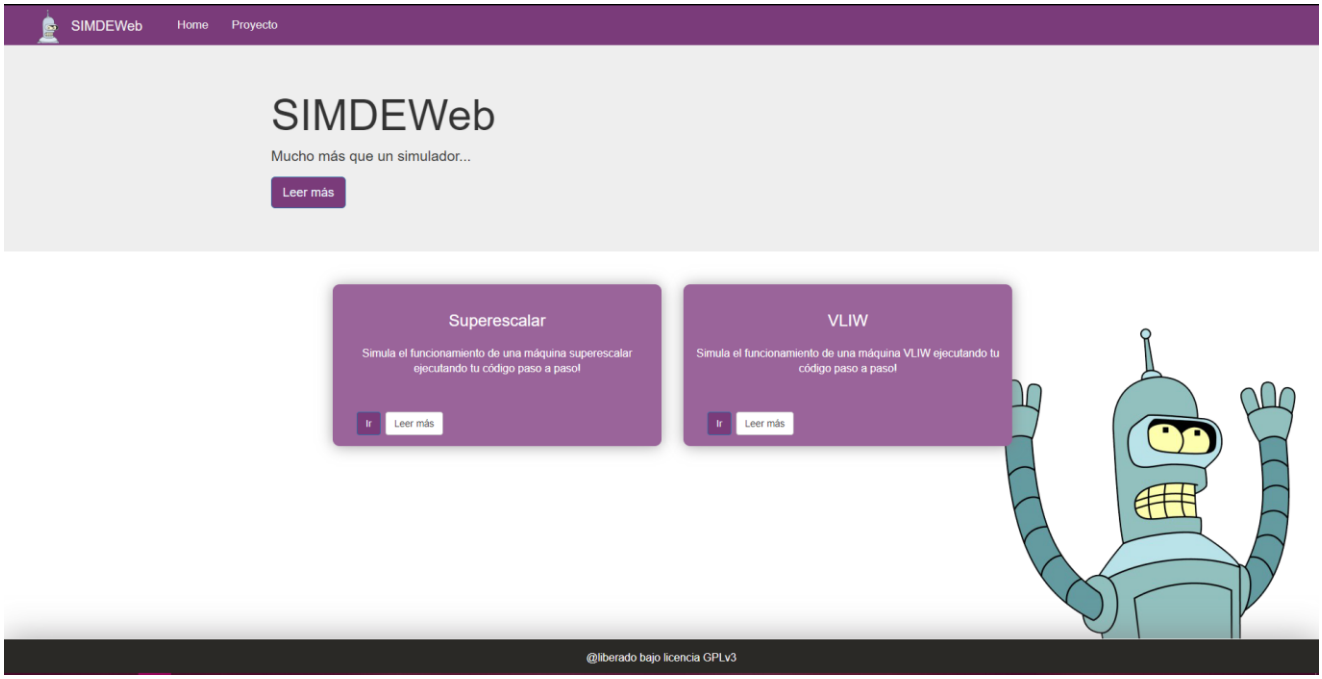


Figura 5:2 Vista inicial SIMDE web

También cuenta con una página *Proyecto* donde se muestra la información principal del proyecto que se encuentra en el GitHub de la aplicación. [5]

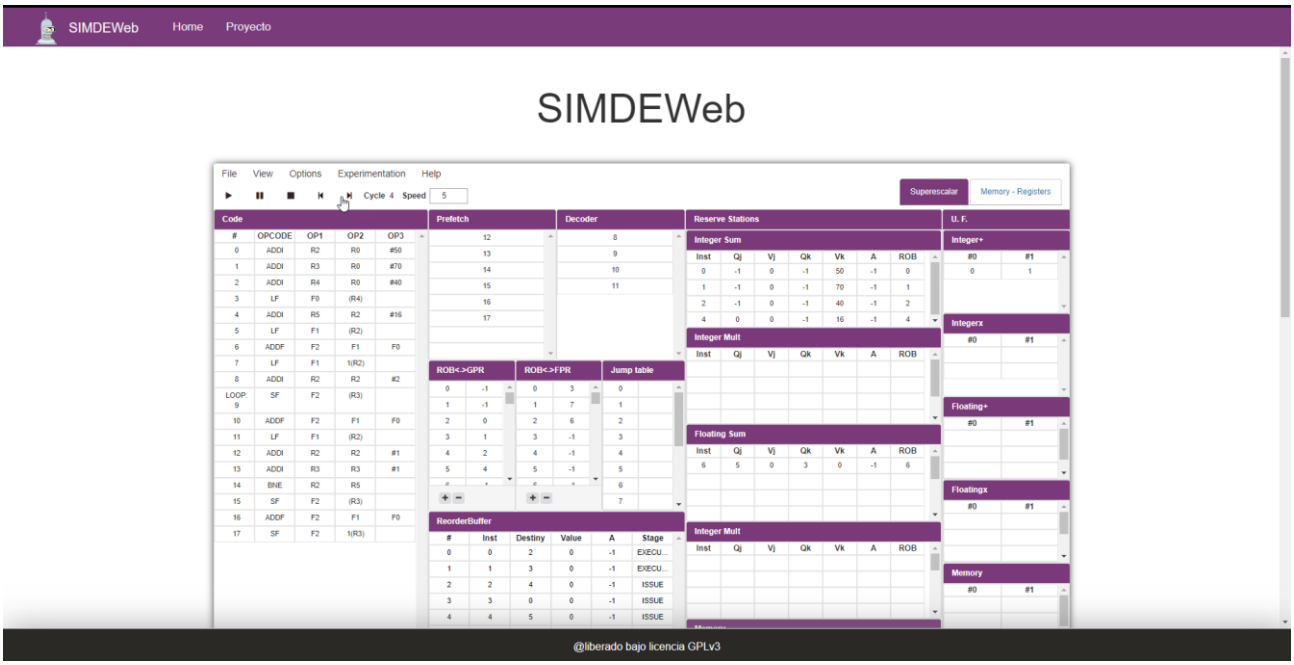


Figura 5:3 Vista proyecto SIMDE web (1)

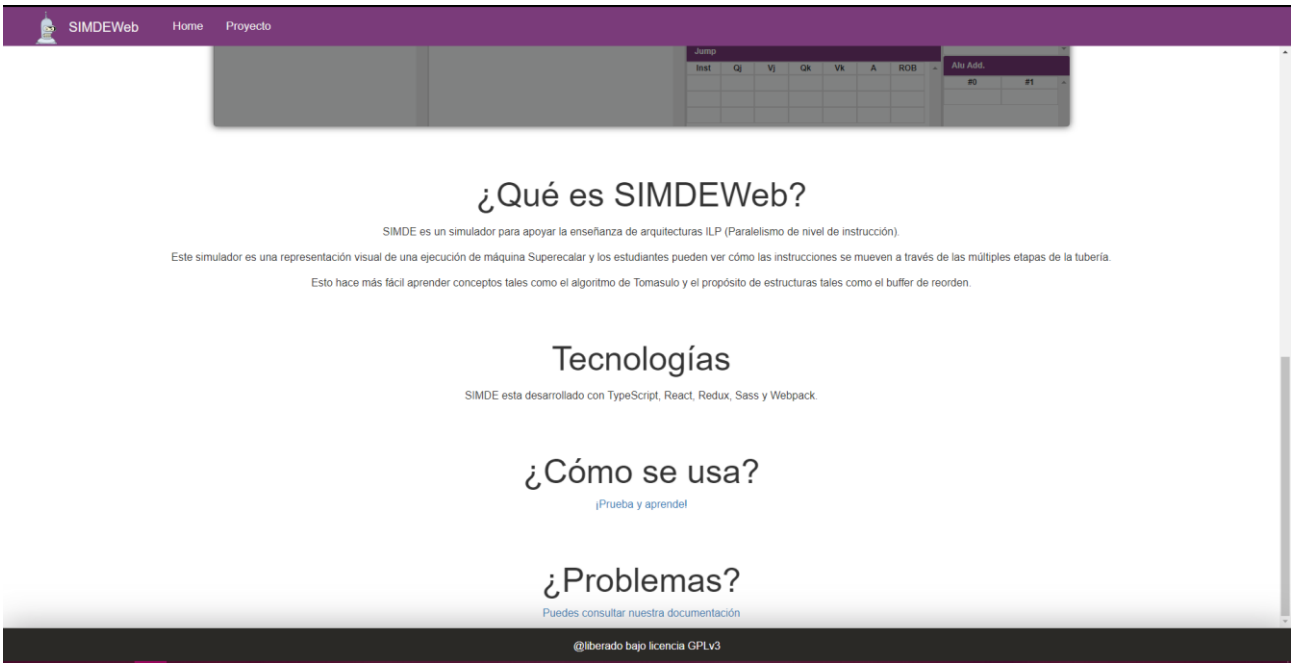


Figura 5:4 Vista proyecto SIMDE web (2)

Esta interfaz fue desarrollada acorde con el estilo de SIMDE web que se encuentra disponible [5] con el fin de mantener una homogeneidad y dar la sensación de una única aplicación de cara al usuario.

5.4 Migración de la máquina VLIW

Antes de comenzar la migración de la máquina VLIW se analizó el código de la versión original de SIMDE [4] y posteriormente se estudió el funcionamiento del código ya desarrollado para SIMDEweb [5]. Esto ayudó a tener una idea de cómo se había afrontado la migración de la máquina superescalar y sirvió como punto de partida en la migración de la máquina VLIW.

A pesar de tener como referencia la máquina superescalar, al ir migrando de C++ a TypeScript se fueron echando en falta algunas características que este último no incluye como serían los punteros, las estructuras, los enums y la sobrecarga de operadores entre otros. Esto obligaba a buscar formas alternativas para simular el funcionamiento del código original presidiendo de esas características que hacen de C++ un lenguaje sólido y completo.

Con el fin de simular estas características, elementos como las estructuras o enums se transformaron en clases lo que hacía la implementación más sencilla. Así mismo, el volumen de código de la aplicación en ocasiones dificultaba identificar y asociar métodos ya implementados a la nueva implementación. Sin embargo, gracias al uso de *intellisense* integrado en Microsoft Visual Studio se pudieron solventar muchos errores sintácticos antes de comenzar a realizar pruebas ahorrando así mucho tiempo.

Tras finalizar la implementación se realizaron una serie de pruebas unitarias para probar algunos métodos y los resultados obtenidos en la ejecución.

5.5 Pruebas Unitarias

Para probar el buen funcionamiento de la máquina VLIW se decidió crear unos archivos de pruebas unitarias que “automatizaran” la comprobación de los resultados que deberían obtenerse según los obtenidos en el SIMDE original.

Las pruebas unitarias o *unit testing* son principalmente trozos de código diseñados para comprobar que el código está funcionando de la forma que esperamos. Se crean pequeños test específicos para las diferentes partes del código con el fin de verificar los resultados. [20] Esta metodología agiliza enormemente el proceso de testeo y si utilización cuenta con muchas ventajas como:

- Calidad del código: Al realizar pruebas continuamente y detectar los errores, al terminar tenemos un código limpio y de calidad.

- Facilita los cambios y favorece la integración: Los *tests* unitarios permiten modificar partes del código sin afectar al conjunto, simplemente para poder solucionar *bugs* que nos encontremos durante la implementación.

- Facilita el proceso *debugging*: Cuando se encuentra un error o *bug* en el código, solo es necesario desglosar el trozo de código testeado para encontrar la procedencia del error. [20]

Para crear las pruebas unitarias solo tenemos que crear un archivo con la extensión .spec.ts en el alojaremos las pruebas que creamos necesarias para nuestro código.

```
    superescalarCode.load(inputSuperescalar);
    code.load(inputVLIW, superescalarCode);
    vliw.code = code;

    while (vliw.tic() !== VLIWError.ENDEXE) { }

    t.deepEqual(vliw.status.cycle, 56, 'Doubleloop: Bad pc at finish');


});
```

Figura 5:5 Ejemplo prueba unitaria

La figura 5:5 muestra un ejemplo de prueba unitaria donde ejecutamos la máquina VLIW introduciéndole un código superescalar juntos con su respectivo código VLIW y se espera que se ejecute en 56 ciclos. Si no es así dará un error y mostrará que el *test* no ha sido superado por línea de comandos. De esta forma podemos ir probando cada una de las funcionalidades del código de manera automática.

5.6 Interfaz de visualización

Al comenzar con el desarrollo de la interfaz se analizaron las decisiones de diseño realizadas en la implementación de la versión actual de SIMDEweb [5]. De esta forma se detectaron los módulos que eran reutilizables y aquellos que necesitaban ser implementados para visualizar el funcionamiento de la máquina VLIW.

5.6.1 Módulos reutilizables

Durante el desarrollo de la primera versión de SIMDEweb se desarrollaron una serie de módulos independientes que permitían visualizar el funcionamiento de la máquina superescalar y que son necesarios también en la máquina VLIW:

- *Access Bar Component*: Corresponde a la barra superior donde se encuentran las diferentes opciones de ejecución de la máquina:

  - *Play*: Ejecutar el código en el simulador.
  - *Pause*: Detener la ejecución en el punto en el que se encuentra.
  - *Stop*: Detener completamente la ejecución.
  - *Back*: Volver a un ciclo de ejecución anterior.
  - *Forward*: Adelantar la ejecución.
  - *Velocidad*: Permite cambiar la velocidad de ejecución de la máquina.

Figura 5:6 Access Bar interfaz VLIW

- *File Bar Component*: Barra de opciones de la parte superior de la aplicación, se encarga de encapsular todas las opciones en diferentes desplegables:

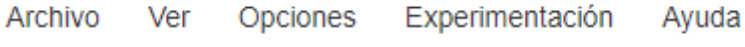


Figura 5:7 File Bar interfaz VLIW

Archivo → Cargar: Permite cargar el código superescalar desde fichero dando la opción de escribir el código directamente en la interfaz pudiendo prescindir de la carga por fichero. Este módulo fue adaptado para poder cargar de la misma forma el código VLIW.

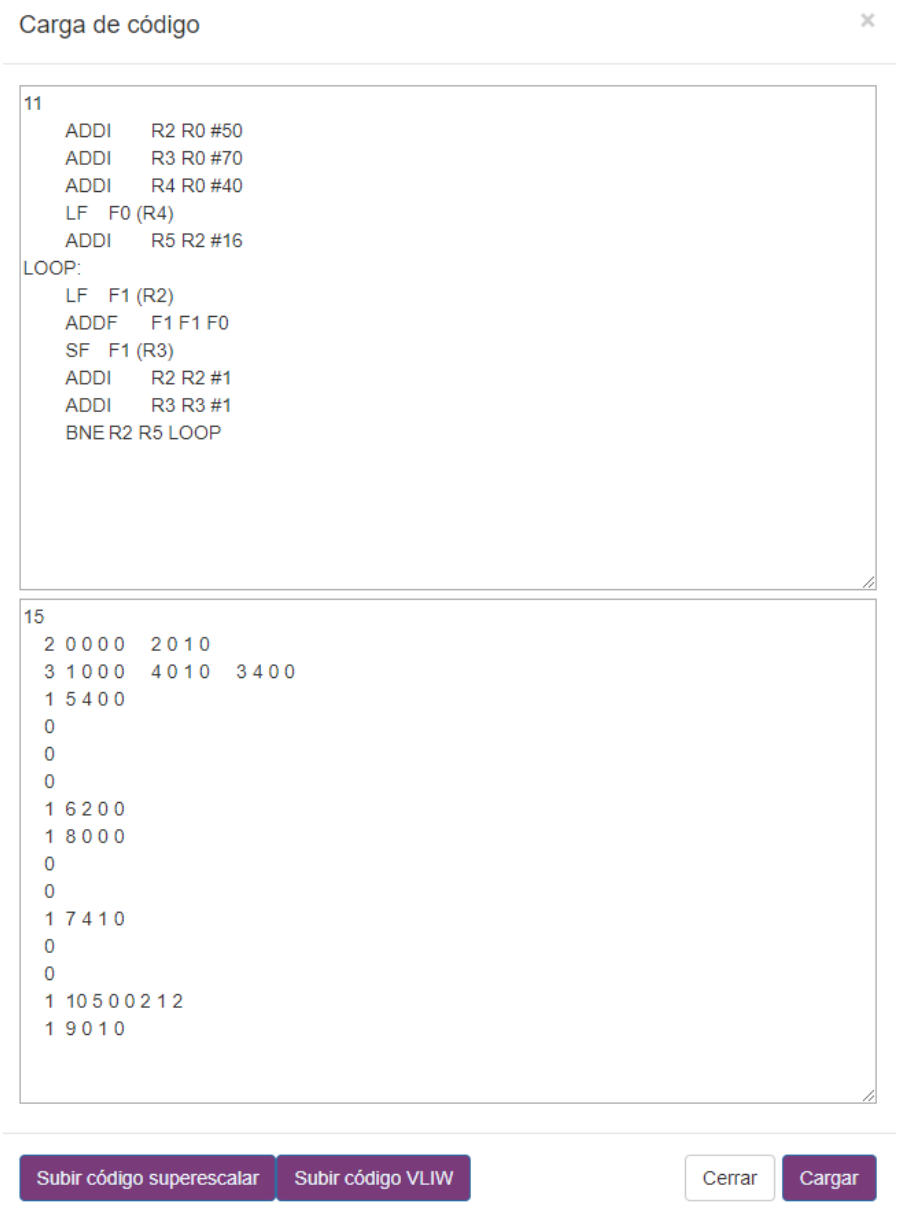
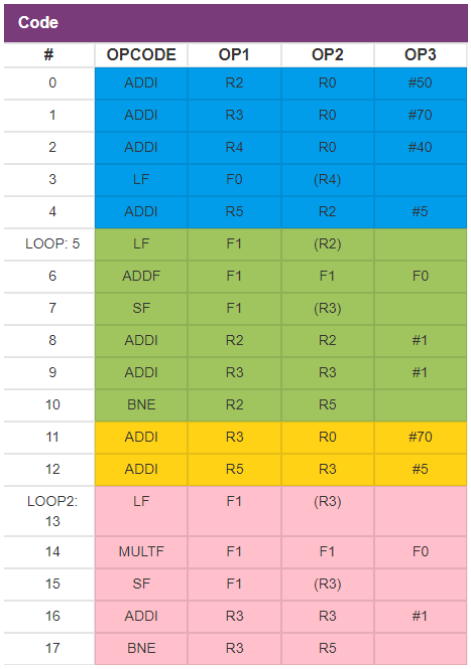


Figura 5:8 Carga código superescalar y VLIW interfaz VLIW

Ver → Bloques básicos: Permite la visualización del código superescalar en la interfaz junto con sus bloques básicos.



Code

#	OPCODE	OP1	OP2	OP3
0	ADDI	R2	R0	#50
1	ADDI	R3	R0	#70
2	ADDI	R4	R0	#40
3	LF	F0	(R4)	
4	ADDI	R5	R2	#5
LOOP: 5	LF	F1	(R2)	
6	ADDF	F1	F1	F0
7	SF	F1	(R3)	
8	ADDI	R2	R2	#1
9	ADDI	R3	R3	#1
10	BNE	R2	R5	
11	ADDI	R3	R0	#70
12	ADDI	R5	R3	#5
LOOP2: 13	LF	F1	(R3)	
14	MULTF	F1	F1	F0
15	SF	F1	(R3)	
16	ADDI	R3	R3	#1
17	BNE	R3	R5	

Figura 5:9 Visualizar código superescalar y bloques básicos interfaz VLIW

Opciones → Configurar máquina VLIW: Permite cambiar la configuración de la máquina VLIW incrementando o disminuyendo el número de unidades funcionales de cada tipo y la latencia de cada una de las unidades funcionales. Este módulo fue modificado para adaptarlo a los parámetros de configuración necesarios en la máquina VLIW.

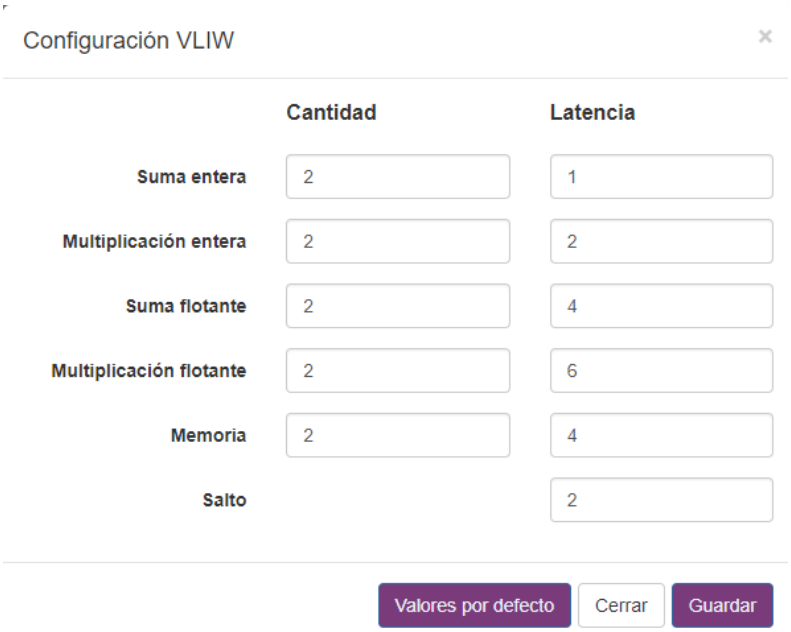


Figura 5:10 Configurar máquina VLIW interfaz VLIW

Opciones → Cargar Memoria - Registros: Permite cargar desde fichero el contenido de la memoria y los registros de la máquina pudiendo escribir directamente en la interfaz prescindiendo así de la carga por fichero.

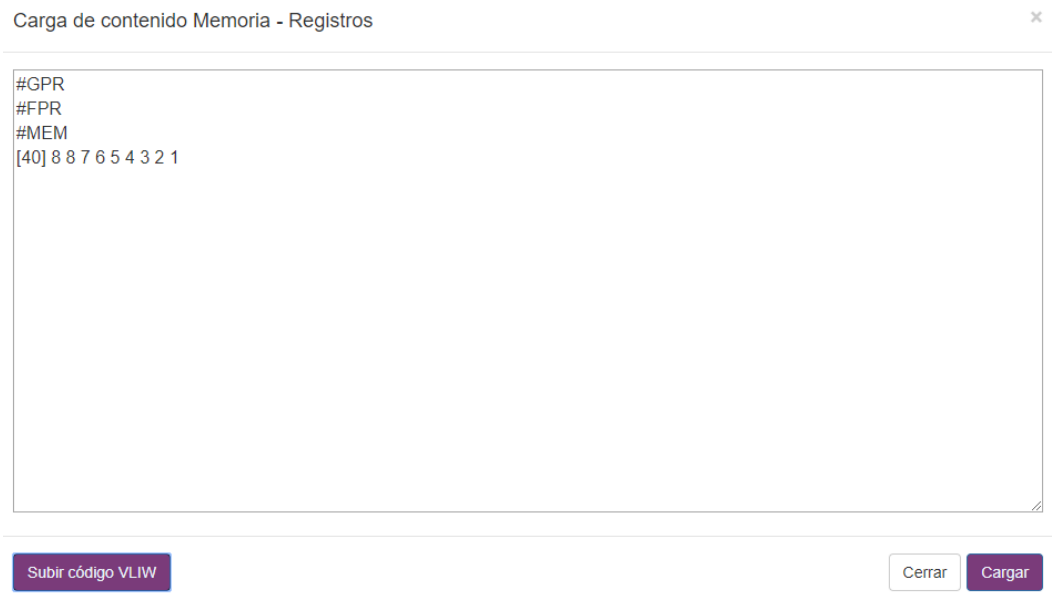


Figura 5:11 Cargar memoria - registro interfaz VLIW

Experimentación → Ejecución por lotes: Permite modificar el porcentaje de fallos de caché y la latencia de dichos fallos. Además, permite configurar la ejecución por lotes.

Figura 5:12 Configurar ejecución por lotes interfaz VLIW

Ayuda →Documentación: Permite acceder a la documentación del simulador.

Ayuda →Acerca de…: Proporciona información sobre los autores de la aplicación y la migración.

- *Functional Unit Component*: Permiten visualizar el estado de las diferentes unidades funcionales que tiene la máquina durante la ejecución.

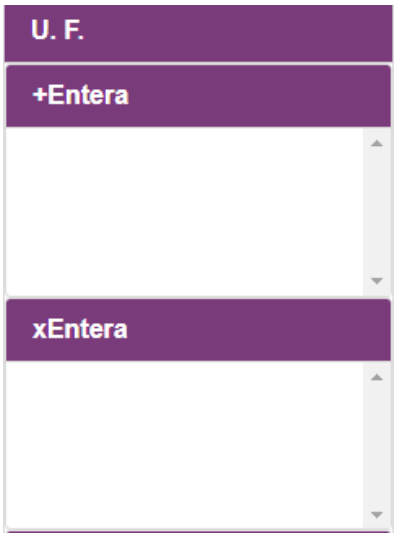


Figura 5:13 Visualizador unidades funcionales interfaz VLIW (recortado)

- *Register Component*: Permite visualizar el estado de los registros/ memoria de la máquina durante la ejecución.

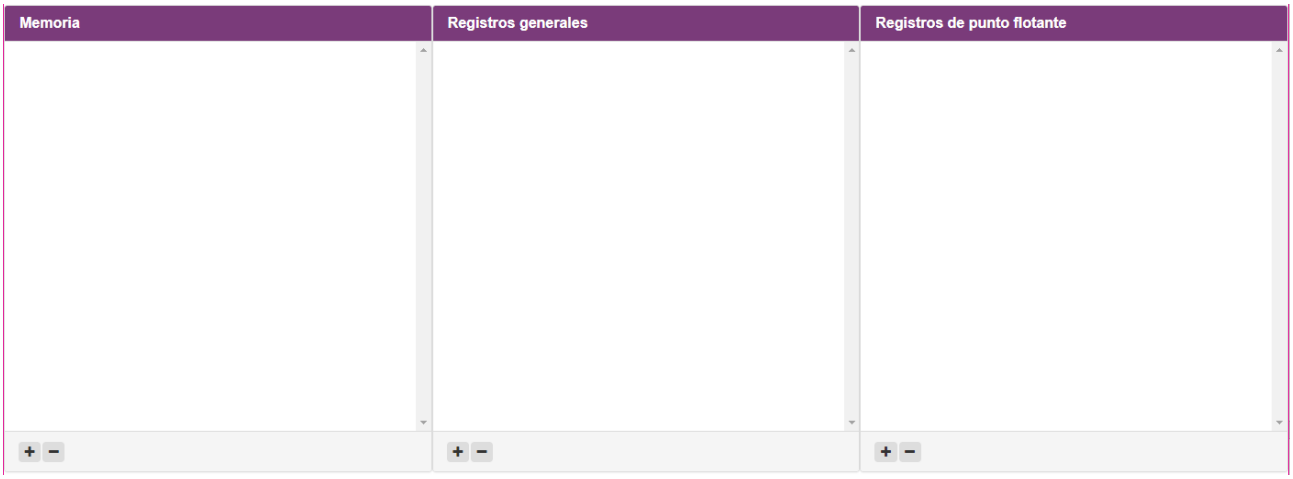
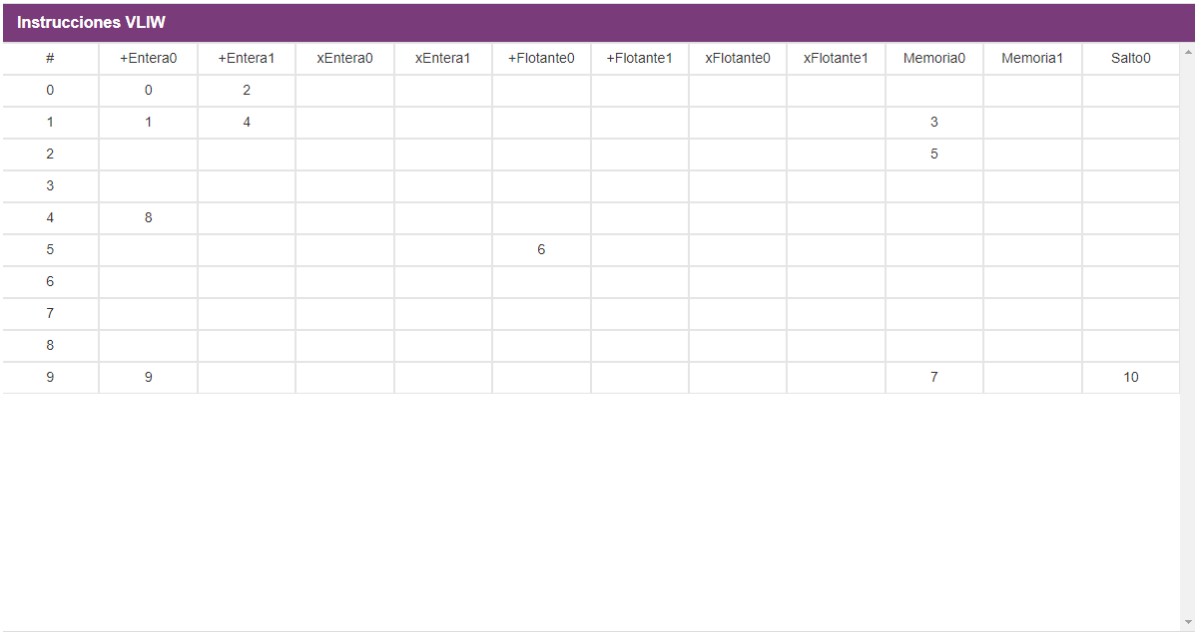


Figura 5:14 Visualizador Memoria- Registros interfaz VLIW

5.6.2 Módulos nuevos

Para la implementación de la maquina VLIW fue necesario desarrollar nuevos módulos que permitieran visualizar el funcionamiento de esta:

- *Table Component*: Permite visualizar las instrucciones largas cargadas desde fichero durante la ejecución.



#	+Entera0	+Entera1	xEntera0	xEntera1	+Flotante0	+Flotante1	xFlotante0	xFlotante1	Memoria0	Memoria1	Salto0
0	0	2									
1	1	4							3		
2									5		
3											
4	8										
5					6						
6											
7											
8											
9	9								7		10

Figura 5:15 Visualizador instrucciones largas interfaz VLIW

- *Register Component*: Este componente se utilizó para la creación de nuevos módulos que compartían características con este:

- NaT (GPR y FPR): Componente de tipo *register* que visualiza el estado del campo Not A Thing utilizando V para verdadero y F para falso.

- Predicado: Componente de tipo *register* que visualiza el estado del predicado utilizando V para verdadero y F para falso.

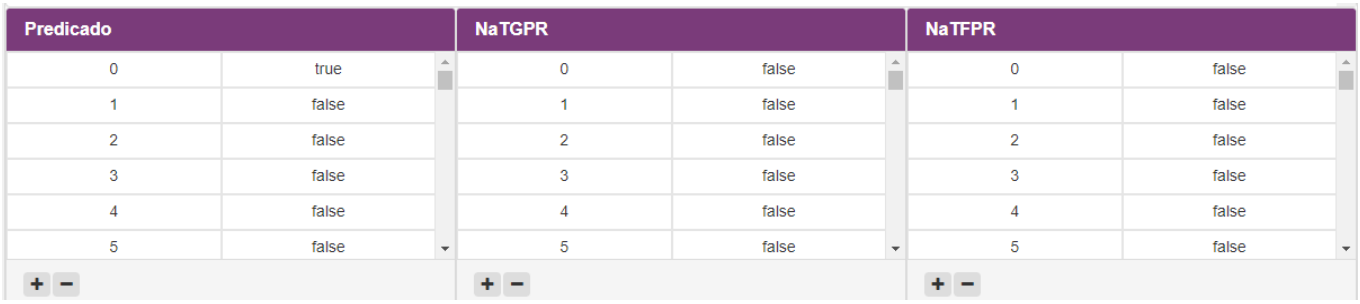


Figura 5:16 Visualizador Predicado y NaT (GPR y FPR) interfaz VLIW

5.6.3 Integración interfaz - core

Para la integración entre la lógica del *core* y la interfaz se ha querido hacer uso de *callbacks* utilizando un sistema de control de estados.

Cuando un componente se renderiza se suscribe a un mismo objeto general que concentra todos los componentes y los pasos se invocan mediante el uso de callbacks. Redux centraliza todas las intenciones de cambio de la interfaz y las gestiona. De esta forma garantizamos que se ejecuten los cambios de uno en uno evitando posibles solapamientos. Posteriormente React toma las actualizaciones de estado y las traduce en una representación virtual de la página resultante. Cada vez que React detecta un cambio vuelve a ejecutar las funciones que realizan cambios en el DOM para reflejar la nueva página.

El resultado es una interfaz donde podemos visualizar el funcionamiento de la máquina VLIW con todas sus características.

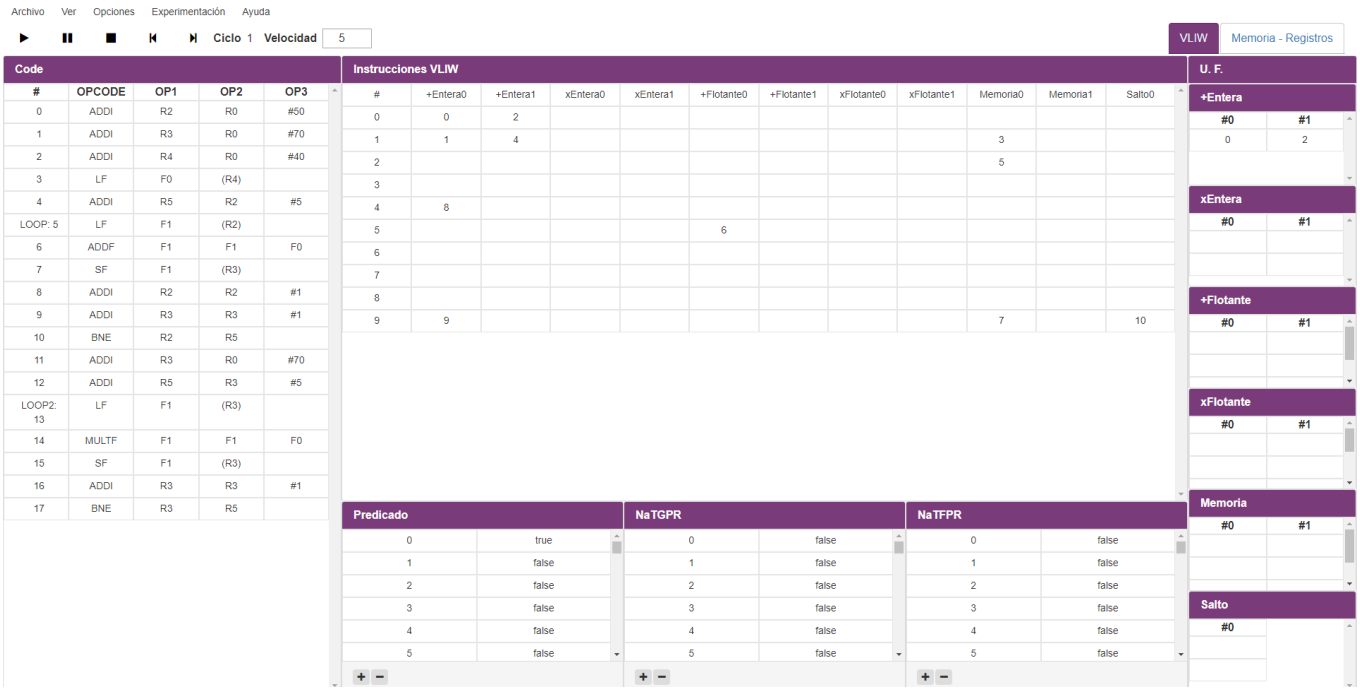


Figura 5:17 Interfaz VLIW SIMDE web

5.7 Migración de documentación

Tras el desarrollo del proyecto se quiso migrar la documentación disponible sobre VLIW integrándola con la ya disponible sobre superescalar. La documentación elaborada para SIMDE original estaba en formato .HLP, este formato de ayuda de Windows quedó obsoleto con Windows Vista lo que impedía abrir el documento en actualizaciones de Windows posteriores.

Esta documentación no solo disponía de datos de la aplicación sino también de una extensa explicación del funcionamiento de cada uno de los componentes necesarios para la máquina Very Long Instruction Word por lo que resultaba muy interesante el poder recuperarla y que pudiera ser usada actualmente en la nueva implementación de la aplicación.

Para poder recuperar la documentación se hizo uso de la herramienta Help Decompiler [21] la cual procesa los ficheros .HLP y genera un fichero .RTF que contiene la información en texto plano. Extrae además las imágenes utilizadas en la realización de la documentación lo que facilitó poder incluirlas en la nueva documentación.

Tras limpiar los archivos de documentación tanto en inglés como en español se utilizó la documentación de VLIW que no se encontraba incluida en documentación de SIMDE web para crear archivos Markdown correspondientes a cada uno de los apartados que se querían agregar.

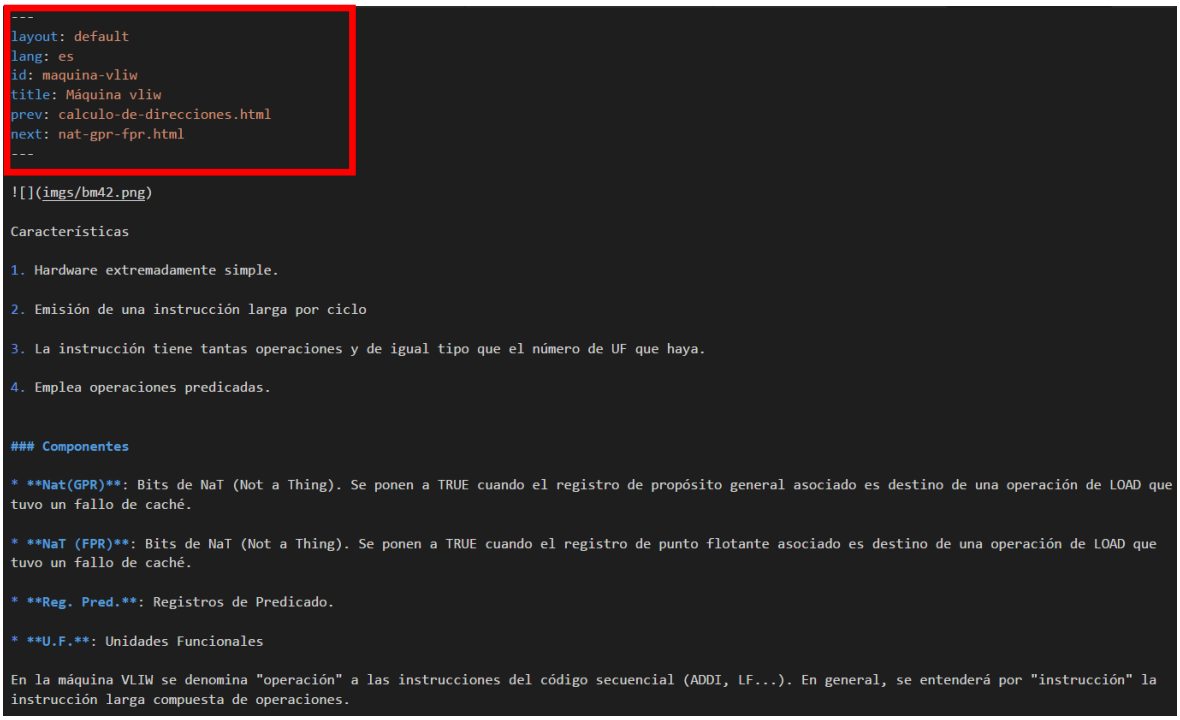


Figura 5:18 Markdown de máquina VLIW

Como se puede ver en la figura 5:18 generar la documentación de esta forma es muy sencillo. Solo es necesario fijar al principio del documento algunos atributos para que posteriormente Gitbook pueda generar la documentación.

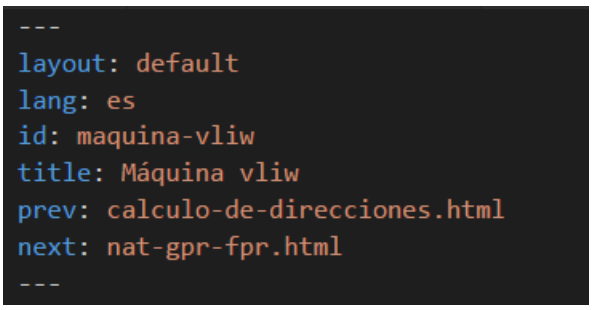


Figura 5:19 Atributos máquina VLIW

Cada uno de los atributos tiene un significado específico que debemos rellenar en cada uno de los Markdown que realicemos:

- *Layout*: especifica el tema que se utilizará para generar la documentación, existe varios, pero en este caso especificamos el que viene por defecto.

- *Lang*: especifica el lenguaje para el que está realizado ese Markdown, esto se utiliza para cuando se quiere generar una documentación en varios idiomas.

- *Id*: un identificador único para distinguir el Markdown.

- *Title*: corresponde al título que queremos darle al Markdown.

- *Prev*: corresponde al Markdown que debe aparecer antes del que estamos realizando.

- *Next*: corresponde al Markdown que debe aparecer después del que estamos realizando.

Finalmente, tras realizar todos los archivos que necesitamos, especificaremos en el archivo SUMMARY.md el título y el nombre de cada archivo de la siguiente forma:

* [titulo](nombre_archivo.md)

Podemos dividir además la documentación en apartados utilizando ###Nombre_apartado.

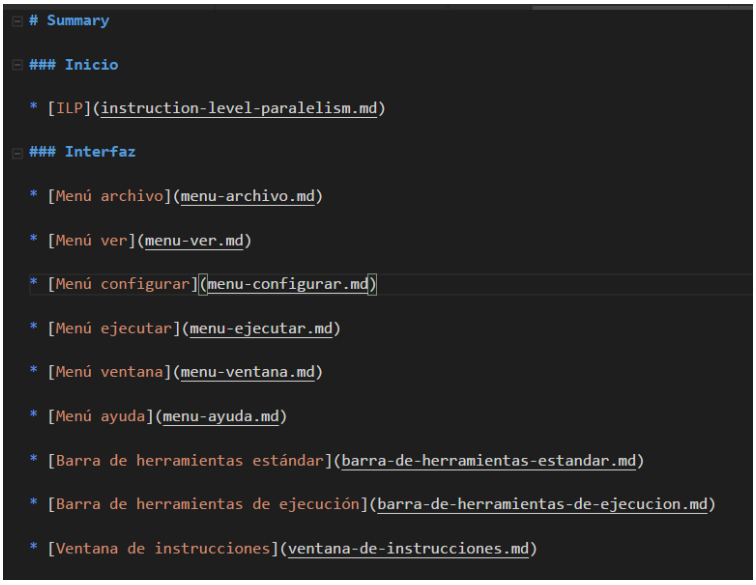


Figura 5:20 Archivo SUMMARY.md

Finalmente generaremos la documentación utilizando la herramienta Gitbook ejecutando en la carpeta raíz del proyecto los siguientes comandos:

$ gitbook install

$ gitbook init

$ gitbook serve

Como resultado obtendremos una documentación como la que se muestra en la siguiente imagen:

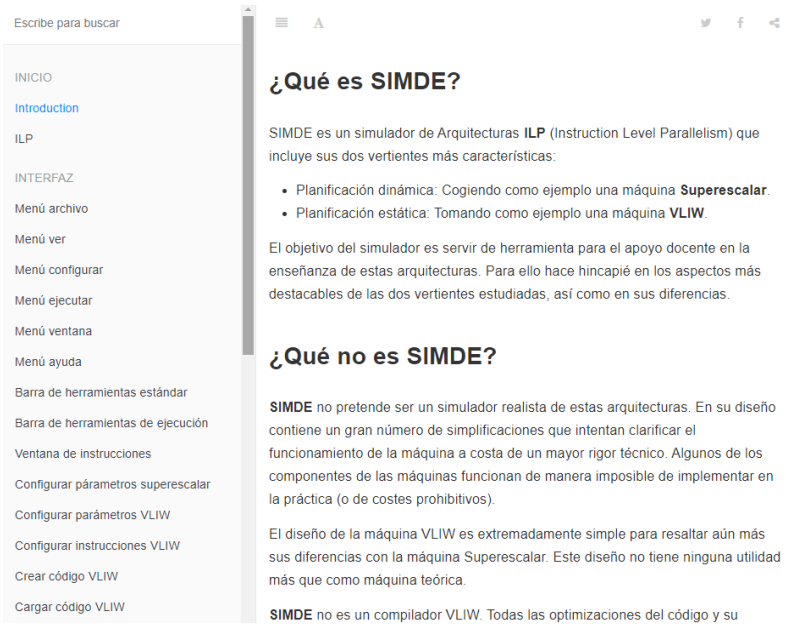


Figura 5:21 Documentación generada

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

Con el desarrollo de este trabajo se ha conseguido disponer de un simulador de la máquina VLIW integrado con SIMDEweb [5] así como una interfaz de integración. Esta interfaz consigue como resultado una aplicación en la que se puede acceder a los simuladores de las máquinas disponibles, así como a la información del proyecto y la documentación desarrollada.

Durante la realización de este proyecto se han adquirido conocimientos sobre tecnologías web y se han integrado buenas prácticas de programación manteniendo la homogeneidad con el código ya desarrollado. Estas características facilitan implementaciones futuras, así como la migración a tecnologías que puedan desarrollarse en un futuro y sean consideradas una mejor alternativa para la implementación del proyecto.

6.2 Líneas futuras

Tras el desarrollo de este trabajo se abren varias líneas futuras:

- Planificación de instrucciones interactiva: Modificar el planificador de instrucciones para que sea interactivo de forma que se pueda planificar las instrucciones largas directamente en el simulador VLIW utilizando alguna librería como React DnD que permita hacer *Drag and Drop* de las instrucciones dentro del planificador.

- Guardar planificación en fichero: Una vez implementada la interfaz interactiva es necesario dar la opción de guardar la planificación realizada en el simulador en un fichero para poder recuperar esta configuración en otra ocasión.

- Nueva interfaz de memoria: Con el fin de poder acceder a la memoria de la máquina sin tener que cambiar de pestaña sería una buena idea encontrar una manera de integrarlo en la página principal del simulador. Podría utilizarse una pestaña desplegable tipo chat de forma que cuando queramos consultar la memoria solo tengamos que desplegar la pestaña sin dejar de visualizar el resto del simulador.

- Tutoriales de funcionamiento: A pesar de que la migración del proyecto aumenta su accesibilidad sería recomendable implementar tutoriales de funcionamiento que expliquen de forma sencilla que muestran cada uno de los módulos de la interfaz y como utilizar el simulador cuando el usuario accede por primera vez con el fin de solventar todas las posibles dudas al comenzar a utilizar la aplicación.

- Login de usuarios: Con el objetivo de mantener un seguimiento del trabajo de los alumnos podría implementarse una gestión de usuarios donde el profesor pudiese dar de alta a sus alumnos. Esto permitiría agregar otras características como:

    - Historial: Mantener un historial de los últimos códigos simulados con el fin de poder visualizar los resultados obtenidos o volver a cargar las planificaciones realizadas sin tener que guardar los archivos directamente en el ordenador manteniendo toda la información en un servidor en la nube.

    - Trabajo colaborativo: Permitir a los usuarios hacer implementaciones de forma conjunta abriendo la posibilidad de debate y el aprendizaje en grupo.

- Gamificación: Implementar juegos que motiven la mejora de los códigos a implementar añadiendo un ranking entre los usuarios.

- Desarrollar más simuladores: La idea principal tanto de SIMDE como de SIMDEweb es ayudar al alumnado a comprender el funcionamiento de las diferentes arquitecturas por lo que podemos afirmar que uno de los objetivos principales es continuar implementando simuladores que permitan enseñar otros conceptos de la arquitectura de computadores como podría ser el paralelismo a nivel de hilo.

Capítulo 7
Summary and Conclusions

7.1 Conclusions

This work has led to the obtention of a VLIW machine Simulator with SIMDEweb, [5]as well as an integration interface. As a result, the application user has access to several simulators, as well as the developed documentation and project information.

During the development of this project knowledge of web technologies have been acquired and I have integrated good programming practices while maintaining the homogeneity with the already developed code. These characteristics facilitate future deployments, as well as migration to technologies that can be developed in the future and are considered a better alternative for the implementation of this project.

7.2 Future work lines

Following the development of this work, future lines of enquiry are available:

- Interactive instruction scheduling: modify instruction Planner to make it interactive so that you can plan long instructions directly in the VLIW Simulator using any library as React DnD that allows Drag and Drop of the instructions within the Scheduler.

- Store scheduling in file: implemented once the interactive interface is necessary to give the option to save the scheduling carried out on the Simulator in a file in order to recover these settings on another occasion.

- New memory interface: in order to be able to access the memory of the machine without having to change the tab would be a good idea to find a way to integrate it into the main page of the Simulator. You could use a tab drop-down type chat in a way that when we want to refer to the only memory we have to deploy tab while viewing the rest of the Simulator.

- Operation tutorials: while the migration of the project increases your accessibility would be advisable to implement tutorials operating that explain in a simple way to show each of the interface modules and how to use the Simulator. When the user agrees for the first time in order to solve all the possible doubts begin to use the application.

- Login of users: in order to keep track of the work of the students could be implemented a user management tool where the teacher could register their students. This would make it possible to add other features like:

- History: keep a record of the latest codes simulated in order to be able to visualize the results or reload the carried out schedules without having to save files directly on the computer holding information in a server in the cloud.

- Collaborative work: allow your users to make deployments jointly opening the possibility of debate and group learning.

- Gamification: Implement games that motivate improvement of codes to be implemented by adding a ranking among users.

- Develop more simulators: the main idea of SIMDE and SIMDEweb is to help students understand the functioning of the different architectures for which we can say that one of the main objectives is to continue to implement simulators that you allow to teach other concepts of the architecture of computers such as thread-level parallelism.

Capítulo 8
Presupuesto

Descripción	Coste por hora	Total
220 horas de trabajo	30€(*)	6600€

Tabla 1 Presupuesto

(*) Nota: El precio por hora incluye el coste de amortización del equipo y la cuota de autónomo.

Capítulo 9
Bibliografía

[1] Arquitecturas Segmentadas Lorenzo Moreno Ruiz
https://campusvirtual.ull.es/1617/mod/resource/view.php?id=172959

[2] Paralelismo a nivel de instrucción - OCW - UC3M

http://ocw.uc3m.es/ingenieria-informatica/arquitectura-de-computadores/materiales/es-m3-intro.pdf/at_download/file

[3] EMISION MULTIPLE DE INSTRUCCIONES ARQUITECTURA

http://viralchap.com/6527562-Emision-multiple-de-instrucciones-arquitectura-avanzada-uciel-cohen.html

[4] Iván Castilla Rodríguez. Simulador didáctico de arquitectura de computadores.

PhD thesis, Universidad de La Laguna, 2004.

[5] Adrián Abreu González. Simulador didáctico de arquitectura de computadores.

https://riull.ull.es/xmlui/handle/915/5125

[6] Simulació d'arquitectures VLIW - DDD – UAB.

https://ddd.uab.cat/pub/trerecpro/2007/hdl_2072_8929/PFCClercMas.pdf

[7] VLIW-DLX Simulator for Educational Purposes – CiteSeerX.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.362.6099&rep=rep1&type=pdf

[8] Simple-VLIW: A fundamental VLIW architectural ... - IEEE Xplore.
https://ieeexplore.ieee.org/document/4675563/

[9] Typescript – JavaScript that scales. https://www.typescriptlang.org/

[10] React - A JavaScript library for building user interfaces. https://reactjs.org/

[11] ¿Cómo funciona React.js? – DevCode. https://devcode.la/blog/como-funciona-reactjs/

[12] Getting Started with Web Accessibility in React – Noteworth.

https://blog.usejournal.com/getting-started-with-web-accessibility-in-react-9e591fdb0d52

[13] Introducción a Redux.js – React & Redux – Medium.

https://medium.com/react-redux/introducci%C3%B3n-a-redux-js-8bdf4fe0751e

[14] Webpack: qué es, para qué sirve y sus ventajas e inconvenientes.

https://www.campusmvp.es/recursos/post/webpack-que-es-para-que-sirve-y-sus-ventajas-e-inconvenientes.aspx

[15] An Introduction to Static Site Generators - David Walsh Blog.

https://davidwalsh.name/introduction-static-site-generators

[16] Generadores de sitios estáticos son una opción para blogs de ... – Susana María Urbano Mateos. https://www.actualidadecommerce.com/generadores-sitios-estaticos-una-opcion-blogs-ecommerce/

[17] Documentation | Hexo. https://hexo.io/docs/index.html

[18] GitBook, crea documentación técnica y libros usando Markdown y Git ... – GENBETA

https://www.genbeta.com/desarrollo/gitbook-crea-documentacion-tecnica-y-libros-usando-markdown-y-git-github-de-forma-flexible

[19] GitBook | Procesadores de Lenguajes - GitHub Pages.

https://ull-esit-pl-1617.github.io/tareas-iniciales-erik-jorge-carlos-ruben/docs/Cap1/GitBook.html

[20] Pruebas unitarias: ventajas y caracteristicas | Apiumhub https://apiumhub.com/es/tech-blog-barcelona/beneficios-de-las-pruebas-unitarias/

[21] Decompile HLP Files with the WinHelp Decompiler – HelpScribble. https://www.helpscribble.com/decompiler.html

Enlaces de aprendizaje (*)

React: Cómo es la estructura y funcionamiento de un componente- Carlos Azaustre

https://carlosazaustre.es/estructura-de-un-componente-en-react/

React Router: Declarative Routing for React.js - React Training

https://reacttraining.com/react-router/

React Native · A framework for building native apps using React

https://facebook.github.io/react-native/

(*) Nota: Estos enlaces, aunque no se han utilizado explícitamente para la redacción de esta memoria si se han utilizado en el desarrollo del proyecto por lo que se ha querido citarlos a modo de referencia para el aprendizaje en futuras mejoras de SIMDE web.