

**Informe Final: Beca de Colaboración**



Autor: Bruno burgos kosmalski

Índice

[1 Introducción 3](#_Toc198553424)

[2 Primeros Pasos 3](#_Toc198553425)

[2.1 ¿Qué es RISC-V? 3](#_Toc198553426)

[2.2 ¿Cómo comenzó la investigación? 5](#_Toc198553427)

[3 Simuladores 6](#_Toc198553428)

[3.1 Selección Inicial 6](#_Toc198553429)

[3.2 Selección Final 12](#_Toc198553430)

[4 Investigación Adicional 13](#_Toc198553431)

[4.1 Compiladores y Trabajo con varios ficheros de código 13](#_Toc198553432)

[4.2 Entrada Salida 13](#_Toc198553433)

[5 Conclusiones de la Beca 13](#_Toc198553434)

[5.1 Resultados de Aprendizaje 13](#_Toc198553435)

[5.2 Conclusiones Personales 14](#_Toc198553436)

[6 Anexos 15](#_Toc198553437)

[6.1 Ripes 15](#_Toc198553438)

[6.1.1 Simulador 15](#_Toc198553439)

[6.1.1.1 Introducción 15](#_Toc198553440)

[6.1.1.2 Interfaz 15](#_Toc198553441)

[6.1.1.3 Memorias Caché 15](#_Toc198553442)

[6.1.1.4 Pipeline 16](#_Toc198553443)

[6.1.1.5 Entrada Salida 17](#_Toc198553444)

[6.1.1.6 Trabajo con el simulador 18](#_Toc198553445)

[6.1.2 Instalación y Ejecución 19](#_Toc198553446)

[6.1.2.1 Linux 19](#_Toc198553447)

[6.1.2.2 Windows 20](#_Toc198553448)

[6.1.3 Aspectos Destacables y Limitaciones 21](#_Toc198553449)

[6.2 RARS 21](#_Toc198553450)

[6.2.1 Simulador 22](#_Toc198553451)

[6.2.1.1 Introducción 22](#_Toc198553452)

[6.2.1.2 Interfaz 22](#_Toc198553453)

[6.2.1.3 Memorias Caché 22](#_Toc198553454)

[6.2.1.4 Pipeline 23](#_Toc198553455)

[6.2.1.5 Entrada Salida 23](#_Toc198553456)

[6.2.1.6 Trabajo con el simulador 24](#_Toc198553457)

[6.2.2 Instalación y Ejecución 24](#_Toc198553458)

[6.2.3 Aspectos Destacables y Limitaciones 25](#_Toc198553459)

[6.3 Jupiter 25](#_Toc198553460)

[6.3.1 Simulador 25](#_Toc198553461)

[6.3.1.1 Introducción 25](#_Toc198553462)

[6.3.1.2 Interfaz 25](#_Toc198553463)

[6.3.1.3 Memorias Caché 26](#_Toc198553464)

[6.3.1.4 Pipeline 26](#_Toc198553465)

[6.3.1.5 Entrada Salida 27](#_Toc198553466)

[6.3.1.6 Trabajo con el simulador 27](#_Toc198553467)

[6.3.2 Instalación y Ejecución 28](#_Toc198553468)

[6.3.3 Aspectos Destacables y Limitaciones 28](#_Toc198553469)

[6.4 RISC-V Venus Simulator 28](#_Toc198553470)

[6.4.1 Simulador 29](#_Toc198553471)

[6.4.1.1 Introducción 29](#_Toc198553472)

[6.4.1.2 Interfaz 29](#_Toc198553473)

[6.4.1.3 Memorias Caché 29](#_Toc198553474)

[6.4.1.4 Pipeline 29](#_Toc198553475)

[6.4.1.5 Entrada Salida 29](#_Toc198553476)

[6.4.1.6 Trabajo con el simulador 29](#_Toc198553477)

[6.4.2 Instalación y Ejecución 29](#_Toc198553478)

[6.4.3 Aspectos Destacables y Limitaciones 30](#_Toc198553479)

[6.5 EmulsiV 30](#_Toc198553480)

[6.5.1 Simulador 30](#_Toc198553481)

[6.5.1.1 Introducción 30](#_Toc198553482)

[6.5.1.2 Interfaz 30](#_Toc198553483)

[6.5.1.3 Memorias Caché 30](#_Toc198553484)

[6.5.1.4 Pipeline 31](#_Toc198553485)

[6.5.1.5 Entrada Salida 31](#_Toc198553486)

[6.5.1.6 Trabajo con el simulador 32](#_Toc198553487)

[6.5.2 Instalación y Ejecución 32](#_Toc198553488)

[6.5.3 Aspectos Destacables y Limitaciones 33](#_Toc198553489)

[6.6 Creator 33](#_Toc198553490)

[6.6.1 Simulador 33](#_Toc198553491)

[6.6.1.1 Introducción 33](#_Toc198553492)

[6.6.1.2 Interfaz 33](#_Toc198553493)

[6.6.1.3 Memorias Caché 34](#_Toc198553494)

[6.6.1.4 Pipeline 34](#_Toc198553495)

[6.6.1.5 Entrada Salida 34](#_Toc198553496)

[6.6.1.6 Trabajo con el simulador 34](#_Toc198553497)

[6.6.2 Instalación y Ejecución 34](#_Toc198553498)

[6.6.3 Aspectos Destacables y Limitaciones 34](#_Toc198553499)

[6.7 WebRISC-V 35](#_Toc198553500)

[6.7.1 Simulador 35](#_Toc198553501)

[6.7.1.1 Introducción 35](#_Toc198553502)

[6.7.1.2 Interfaz 35](#_Toc198553503)

[6.7.1.3 Memorias Caché 35](#_Toc198553504)

[6.7.1.4 Pipeline 35](#_Toc198553505)

[6.7.1.5 Entrada Salida 36](#_Toc198553506)

[6.7.1.6 Trabajo con el simulador 36](#_Toc198553507)

[6.7.2 Instalación y Ejecución 36](#_Toc198553508)

[6.7.3 Aspectos Destacables y Limitaciones 36](#_Toc198553509)

[7 Referencias 38](#_Toc198553510)

# 1 Introducción

Este se trata de un documento en el que se describirá todo el proceso de la beca de colaboración con el departamento de arquitectura de la facultad de informática (DATSI), durante el curso 24/25. Esta beca de colaboración ha nacido con el propósito de conocer más a fondo el estándar RISC-V, actualmente en boca de muchos investigadores y universidades como la esperanza contra el monopolio estadounidense.

A lo largo del documento se expondrán los pasos que se han dado en la beca, junto a algunos de los conocimientos adquiridos en esta área.

# 2 Primeros Pasos

Este apartado versará sobre los primeros pasos y decisiones que se tomaron en la beca. Los inicios y primeras expectativas que se tenían sobre la misma.

## 2.1 ¿Qué es RISC-V?

Para entender el sentido de esta investigación lo primero que se debe de hacer es explicar el estándar y su contexto.

RISC-V [1] es un ISA (Instruction Set Architecture) que se encuentra disponible para el libre desarrollo del mismo (open source). Este estándar nace a principios de 2010, en la universidad de Berkeley (California) como un proyecto de investigación con fines inicialmente educativos. Sin embargo, con el paso de los años, este estándar ha ido creciendo en popularidad, en gran parte gracias a la libertad que ofrece en comparación a otras arquitecturas de la competencia como pueden ser x86 o ARM. Esto le está convirtiendo en un futuro pilar dentro del mundo de la electrónica y diseño, siendo que potencias tecnológicas como China y Europa están apostando muy fuertemente por este. Sin embargo, como podemos ver con la actualización de versiones de cada año, este estándar todavía se puede considerar inmaduro, viéndose todavía cambios significativos de versión en versión, donde todavía cuesta ver implementaciones fuera del ámbito de los sistemas empotrados.

Y aún con esto en mente, RISC-V está demostrando ser una apuesta segura para empresas de renombre como puede ser NVIDIA, que ya ha desarrollado más de veinte extensiones para este estándar y cuyas gráficas ya cuentan con una serie de núcleos RISC-V para ayudar con el flujo de datos [2]. Y otras como SiFive, compañía que nació en 2015 con el objetivo de crecer impulsando el estándar [3], contando ahora con varias implementaciones actuales [4] y cierto prestigio y renombre dentro del sector.

Antes se han mencionado las extensiones, y es que este estándar se define por sus extensiones. Las extensiones se pueden entender como ampliación de una base inicial, es decir, como un subconjunto de instrucciones junto a una serie de características hardware. Por ejemplo, la extensión “F” es la que añade las instrucciones de coma flotante en simple precisión (32 bits), y para poder implementar esta extensión se necesitarán registros de coma flotante. Luego, por otra parte, contamos con las bases: estas se diferencian principalmente por el tamaño de los registros y la cantidad de los mismos. Principalmente podemos encontrar cuatro bases definidas: *RV32I*, *RV32E*, *RV64I* y *RV64E*, donde las versiones 32 cuentan con registros de propósito general de 32 bits mientras que las versiones 64 cuentan con registros de 64 bits. También se diferencia entre las versiones *I* y *E*, siendo las versiones *I* las versiones por defecto, con 32 registros de propósito general a diferencia de las *E* que solo cuentan con la mitad de registros de propósito general para ahorrar espacio. Entonces, para definir una implementación del estándar tenemos que centrarnos en la base y extensiones que se definen y se montan sobre la misma.

Precisamente este concepto de las extensiones es lo que dota al estándar de la libertad de diseño que contempla, esto junto a lo que se había mencionado con anterioridad: el hecho de que todo el desarrollo del estándar esté disponible y visible. A continuación, se mencionarán algunas de las extensiones más generales del estándar y sus instrucciones:

* Extensión base (“*I*”) : Esta es la extensión base del estándar, contiene todas las instrucciones básicas para el trabajo con números enteros. Es decir: operaciones aritméticas como la suma y la resta, operaciones lógicas como *AND*, *OR*, *XOR*; operaciones para los accesos a memoria, load y store; operaciones de comparación, operaciones para los saltos condicionales e incondicionales, instrucciones para los desplazamientos aritméticos y lógicos, etc.
* Extensión *“M”*: Esta extensión contiene las instrucciones asociadas a la multiplicación y división de enteros, incluyendo aquellas como el resto.
* Extensión *“A”*: Esta extensión contiene instrucciones que se ejecutarán de forma atómica. Mismas instrucciones aritméticas, lógicas y de carga y almacenamiento en memoria, pero que garantizan una ejecución atómica.
* Extensiones *“F”, “D”, “Q*”: Estas extensiones implementan el trabajo con números de coma flotante: *“F”* para simple precisión, *“D”* para doble precisión y *“Q”* para cuádruple precisión. Donde las instrucciones contenidas son aquellas para las operaciones aritméticas, de carga y almacenamiento en memoria, comparación, etc. Para cumplimentar con todo el trabajo con números de coma flotante.
* Extensión Zcsr: Esta es una extensión fundamental para cualquier implementación con varios niveles de privilegios. Es esta están contenidas las instrucciones que permiten interactuar de forma atómica con los distintos registros de control.

Adicionalmente, se contemplan muchas más instrucciones dentro del estándar, ya no solo las anteriormente mencionadas, sino que ya hay muchísimas implementaciones y algunas que están por fuera del estándar, como lo que se había mencionado antes de NVIDIA.

Por otro lado, en cuanto a lo que se refiere al ensamblador en sí mismo, podemos ver un ensamblador RISC clásico de 3 objetivos, con cinco tipos principales de códigos de operación donde podríamos destacar alguna particularidad: el desbordamiento se tiene que gestionar por software, es decir no hay instrucciones específicas para las operaciones sin signo; en los accesos a memoria, instrucciones load y store, solo se permite un registro base con desplazamiento relativo inmediato, es decir solo se usan dos registros, el registro fuente y el registro base donde se almacena el puntero, y otras características que se pondrán en contraposición en los siguientes apartados.

Escala de tiempo

Descripción generada automáticamenteFigura 1.1

## 2.2 ¿Cómo comenzó la investigación?

Inicialmente la beca se plateó como una comparación entre el ensamblador que está en uso actualmente en la asignatura “Estructura de Computadores” en el grado de Ingeniería Informática: Motorola 88110, ensamblador RISC; y un ensamblador RISC mucho más moderno: RISC-V. Sin embargo, a lo largo de la beca el objetivo se ha ido centralizando en el estándar RISC-V y su entorno, como pueden ser por ejemplo sus diferentes simuladores.

Para cumplir con este objetivo lo primero que se realizó en la beca fue una exploración inicial del estándar RISC-V y su contexto: principales características e implementaciones, instrucciones, diferencias iniciales, etc.

Para destacar las diferencias entre los dos lenguajes ensamblador se hizo una comparativa y traducción instrucción por instrucción entre el simulador del procesador 88110 y las instrucciones declaradas dentro del estándar RISC-V. En este caso se pudieron destacar muchas similitudes entre ambos simuladores considerando el paso de un lenguaje a otro relativamente sencillo. Encontrando apenas algunas diferencias como por ejemplo: tamaño de los datos inmediatos, 4 bits por detrás en el nuevo estándar; en el tratamiento del desbordamiento en las operaciones aritméticas, el nuevo estándar mantiene el concepto de la gestión manual del software del desbordamiento, por lo que no cuenta con ninguna de las instrucciones de operación sin signo; los accesos a memoria se hacen siempre con un desplazamiento inmediato de 12 bits a diferencia del 88110 donde el desplazamiento está contenido en un registro; supresión por el nuevo estándar de las instrucciones innecesarias como por ejemplo la instrucción resta con datos inmediatos, que es equivalente a la suma con el número invertido en signo; un entorno más desarrollado tanto en el uso de los registros que aunque muy similar parecen tener funciones y apodos específicos, y el desarrollo de las pseudoinstrucciones.

Tras la investigación inicial del nuevo estándar (RISC-V), y sus diferencias con respecto al ensamblador del simulador del procesador 88110. Se decidió probar con un proyecto más práctico para reafirmar las mismas y ver si realmente podrían tener un impacto significativo en el desarrollo de una práctica más compleja. Para esto se seleccionaron algunos simuladores web del nuevo estándar para poder probar el código y se hizo una traducción adaptada de la práctica ensamblador que se desarrolla en la asignatura *“Estructura de Computadores”*. Una vez probados las distintas funciones y comprobado su correcto funcionamiento se sacó como conclusión que no solo es una práctica sencilla la adaptación de un código a otro, sino que en la mayoría de casos se ganaba en claridad del lenguaje y se reducía el tamaño del código para desarrollar las mismas funciones.

# 3 Simuladores

Una vez investigado el concepto y contexto del estándar, el siguiente paso en la beca fue buscar un simulador disponible que se adaptara a las necesidades de las asignaturas competentes, en este caso: *“Arquitectura de Computadores”* y *“Estructura de Computadores”*. Centrando la búsqueda en la sencillez del proceso de instalación, la comodidad de la interfaz, la implementación de características como memorias caché, ejecución segmentada (pipeline), entrada salida, etc.

De esta maneara comenzó la búsqueda inicial de posibles candidatos para este puesto.

## 3.1 Selección Inicial

Como primeros candidatos en la búsqueda se seleccionaron los simuladores más populares dentro del entorno, independientemente de la interfaz y sus características. Sin embargo, sí que se descalificó aquellos simuladores cuyo proceso de instalación se consideró demasiado complejo, casos como *“SPIKE”* [5]; aquel hardware propietario y aquellos que tratan de imitar un entorno hardware, casi como máquinas virtuales. En este último caso podemos poner como ejemplo al entorno *“QEMU”.*

Esto llevó a la selección inicial de los siguientes ocho simuladores:

1. [RIPES](#_6.1_Ripes)
2. [RARS](#_6.2_RARS)
3. [Jupiter](#_6.3_Jupiter)
4. [RISC-V Venus Simulator](#_6.4_RISC-V_Venus)
5. [EmulsiV](#_6.5_EmulsiV)
6. [Creator](#_6.6_Creator)
7. [WebRISC-V](#_6.7_WebRISC-V)
8. Eclipse RISCV

Para más información sobre cada uno de estos simuladores véase el apartado [anexo](#_6_Anexos) específico de cada simulador.

Tras la selección inicial se comenzó un proceso de análisis en más profundidad de las características que ofrece cada simulador. Desde los puntos más característicos y destacables, hasta las limitaciones y errores. En este caso algunos de los apartados centrales fueron las características hardware implementadas, el entorno, la interfaz, la comodidad del trabajo con el mismo, etc.

La siguiente figura muestra una tabla resumen de las diferentes características de los diferentes simuladores.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Comparación por Funciones del Simulador** | | | | | | |
| Simulador | Entrada Salida | Pipeline | Simulación de Caché | Documentación | Self-modifying Code | Asignaturas |
| Ripes | MMIO / Sin Excepciones / Sin interrupciones | Gráfico / Deshabilitable Parcialmente | Grafica / Configurable | Disponible / Calidad | No | Arquitectura/Estructura |
| RARS | MMIO / Excepciones / Interrupciones | No | Grafica / Configurable | Disponible / Calidad | Habilitable | Arquitectura/Estructura |
| Jupiter | No | No | Grafica / Configurable | Disponible / Buena | Habilitable | Entructura |
| RISC-V Venus Simulator | ecall / Sin Excepciones / Sin Interrupciones | No | No | Disponible / Suficiente | No | Estructura |
| EmulsiV | Sí / Sin Excepciones / Interrupciones | Gráfico / No deshabilitable | No | Disponible / Buena | No | Arquitectura/Estructura |
| Creator | ecall / Sin Excepciones / Sin Interrupciones | No | No | Disponible / Suficiente | No | Estructura |
| WebRISC-V | No | Gráfico / No deshabilitable | No | Disponible / Suficiente | No | Estructura |
| Eclipse RISCV | ecall / Sin Excepciones / Sin Interrupciones | No | No | Disponible / Suficiente | No | Estructura |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **Comparación por Características del Entorno** | | | | | | |
| Simulador | Proceso de Instalación | Lenguaje programación | Entorno | Sistemas Operativos | Software Libre | Fuentes Disponibles |
| Ripes | Secillo | C++ | Aplicación | Windows/Linux/Mac | MIT License | Sí |
| RARS | Muy sencillo | Java | Aplicación | Windows/Linux/Mac | MIT License | Sí |
| Jupiter | Muy sencillo | Java | Aplicación | Windows/Linux/Mac | GPL-3.0 license | Sí |
| RISC-V Venus Simulator | Muy sencillo | TypeScript/HTML | MS Visual Studio Code | Windows/Linux/Mac | MIT License | Sí |
| EmulsiV | Medio/Ninguno | JavaScript | Web | Windows/Linux/Mac | MPL-2.0 license | Sí |
| Creator | Ninguno | JavaScript | Web | Windows/Linux/Mac | LGPL-2.0 license | Sí |
| WebRISC-V | Ninguno | PHP | Web | Windows/Linux/Mac | BSD 3-Clause License | Sí |
| Eclipse RISCV | Complejo | C/C++ | Eclipse Embedded | Windows/Linux/Mac | EPL-2.0 license | Si |

A continuación, se añaden unas tablas con las principales ventajas y desventajas de cada simulador.

* Ripes:

|  |  |
| --- | --- |
| **RIPES** | |
| Ventajas: |  |
| - Extensión de la implementación: Este simulador es uno de los más completos en su implementación, contando con soporte para varios modos de ejecución, implementadas las principales extensiones, junto a características interesantes como puede ser la ejecución segmentada, memorias caché, entrada salida, etc. | |
|  |
| - Calidad de la documentación: La documentación disponible es de fácil acceso y detalla la mayor parte de la implementación y principales funciones del simulador | |  |
| - Interfáz Gráfca: La interfaz del simulador es muy gráfica y detalla en muchos apartados, por ejemplo cuando se trabaja con las memoria cachés. Por otro lado, el proceso de trabajo con este simulador es realmente cómodo en comparación a otros del estilo: tanto para el código como para la depuración es relativamente intuitivo, incluso para un usuario que acaba de empezar. | |  |
|  |
| - Trabajo con programas más complejos: Este simulador permite la ejecución de programas ejecutables en el formato, lo que implica que contando con el compilador adecuado se puede trabajar con programas distribuidos en varios ficheros, o incluso en varios lenguajes como C y ensamblador. Por otra parte, también se nos permite un poco más de configuración en los espacios de memoria gracias a los ficheros de enlace, aunque con algunas limitaciones en comparación a un entorno real. | |  |
|  |
|  |
| Desventajas: |  |  |
| - Lenguaje y Formato del programa: Para ampliar la implementación de este simulador hay que tener en cuenta el lenguaje en que está escrito. En este caso nos encontramos con un programa escrito en c++ pero con una similación de entorno, prácticamente hardware, lo que vuelve esta una programación bastante compleja y difícil de entender. | |  |
|  |
| - Limitaciones en la entrada salida: Este simulador aunque cuenta con una implementación de algunos periféricos, no cuenta con soporte para interrupciones ni excepciones, es decir, toda la entrada salida es programada. Correspondientemente no cuenta con una implementación de los niveles de privilegios del estándar, registros de control o instrucciones que puedan trabajar con ellos. | |  |
|  |

* RARS:

|  |  |
| --- | --- |
| **RARS** | |
| Ventajas: |  |
| - Extensión de la implementación: Este simulador cuenta con una implementación muy completa de las principales y más interesantes caracrterísticas, como puede ser la entrada salida, las memorias caché; esto junto con la mayor parte de las instrucciones del estándar, al menos de las principales extensiones. | |
|  |
| - Lenguaje: Este programa está escrito en java, donde solo se necistia una versión del jdk superior o igual a la 8. Esto, junto a la simplicidad en el planteamiento del simulador, vuelven relativamente sencilla el ampliar la implementación en caso de ser necesario. También reduce la complejidad para el despliegue e instalación del propio simulador. | |  |
|  |
| - Documentación: La documentación que se encuentra disponible de las funciones del simulador es bastante completa, e incluso la documentación de la implementación puede ser suficiente. | |  |
| - Sistema de interrupciones y excepciones: Este simulador, cuenta con soporte para interrupciones y excepciones en su implementación, cosa que raramente se encuentra en otros simuladores. | |  |
| Desventajas: |  |  |
| - Interfaz gráfica: La interfaz gráfica del simulador en algunos casos deja mucho que desear: se pueden encontrar algunos bugs visuales en la implementación, junto a algunos apartados en los que no es suficiente. | |  |
|  |
| - Versión de la implementación: En algunos puntos de este programa podemos encontrar algunos apartados que se han quedado obsoletos con respoecto a la versión actual del estándar, por ejemplo con la entrada salida y los registros de contrón, donde la última actualización de este simulador data de principios de 2020. | |  |
|  |
| - Programación y depuración: El trabajo de programación y depuración con el código puede parecer un poco tedioso en algunos puntos: cuando se ejecuta el código no se pueden ver las etiquetas, y la edición del código se encuentra en otro apartado. | |  |
|  |

* Jupiter:

|  |  |
| --- | --- |
| **Jupiter** | |
| Ventajas: |  |
| - La interfaz de este simulador: es realmente sencilla, lo que permite adaptarse a ella muy fácilmente en comparación a otros del entorno. | |
| - El proceso de instalación y configuración: muy sencillo, donde el único prerrequisito necesario es tener una versión del JDK 8 o superior instalada. | |
| - El lenguaje en el que está escrito el simulador: escrito en java, es relativamente amigable, facilitando la posibilidad de añadir cualquier tipo de funcionalidad o de arreglar cualquier tipo de desperfecto. | |
| Desventajas: |  |
| - No cuenta con apatados interesantes: como puede ser la entrada salida o la ejecución segmentada. | |
| - La implementación del estándar: muy limitada, por ejemplo no permite la declaración de macros. | |
| - Trabajo con el simulador: se ouede volver bastante incómodo en el apartado de ejecución y depuración de código, por el formato de la interfáz de ejecución. | |
| - Presencia en el entorno: este simulador no ofrece nada particular o que lo diferencie del resto de simuladores de su entorno, precisamente por su sencillez en la implementación no llega a destacar. | |

* RISC-V Venus Simulator:

|  |  |
| --- | --- |
| **RISC-V Venus Simulator** | |
| Ventajas: |  |
| Como la principal ventaja de este simulador, tenemos que este simulador se despliega sobre un entorno con el que los alumnos están bastante familiarizados como es Visual Studio Code. Donde el proceso de instalación y configuración es relativamente sencillo y se puede hacer directamente desde el propio entorno. Es fácil de configurar y preparar, contando también con algunos periféricos ya dados para poder jugar con la entrada salida, aunque esta no sea mediante interrupciones. | |
|  |
|  |
|  |
| Desventajas: |  |  |
| Para resaltar las principales desventajas de este simulador: tenemos que este simulador puede quedarse escaso en ciertos aspectos, por ejemplo, no cuenta con simulación de memorias cahcé, ni con un pipeline gráfico, juntando que la entrada salida se hace enteramente mediante llamadas, no por interrupciones. Adicionalmente la visualización del valor de los registros y la memoria es un tanto simple, suficiente, pero mejorable. Por otro lado, dada la naturaleza del entorno, cambiar el código fuente o añadir nuevas funcionalidades puede convertirse en una tarea bastante compleja. | |  |
|  |
|  |
|  |

* EmulsiV:

|  |  |
| --- | --- |
| **EmulsiV** | |
| Ventajas: |  |
| Como principal ventaja de este simulador, tenemos que se despliega sobre un entorno web, lo que reduce significativamente la instalación y configuración previas necesarias. Por otro lado, tenemos que este simulador cuenta con simulación de entrada salida mediante interrupciones, de forma nativa. También cuenta con un apartado gráfico y dinámico para el pipeline y su ejecución. | |
|  |
|  |
|  |
| Desventajas: |  |  |
| Como principal desventaja de este simulador, nos encontramos con su entorno: a pesar de no necesitar mucho para la instalación, el que se ejecute sobre otro entorno web genera otra serie de problemas, como por ejemplo el donde se alojará la propia página, o si podría haber problemas de sobresaturación. Por otro lado en la interfáz propia del simulador, el pipeline y sus características no se pueden deshabilitar en la versión fuente, donde cabe también añadir que no cuenta con simulación de memoria caché. | |  |
|  |
|  |
|  |

* Creator:

|  |  |
| --- | --- |
| **Creator** | |
| Ventajas: |  |
| Como ventaja a destacar, este simulador se ejcuta en un entorno web por lo que no es necesario ningún proceso previo de instalación o configuración. También tenemos un entorno muy sencillo de entender y ya configurado para poder escribir y ejecutar código ensamblador riscv. Perfecto para cuando solo se necesita hacer pequeñas pruebas, o cuando recién se está empezando. | |
|  |
|  |
|  |
| Desventajas: |  |  |
| Como principal desventaja tenemos que, este simulador ejecuta sobre un entorno web lo que ya hemos comprobado que puede ser un problema en determindos casos: por ejemplo, que se pueda sobresaturar el entorno. Por otro lado, si se busca algo más complejo, como por ejemplo entrada salida, o simulación de cachés no cuenta con estos apartados. Añadiendo a esto, podemos considerar que este simulador quizá se pase de simple, teniendo muy pocas opciones de configuración, donde la mayoría son meramente estéticas. | |  |
|  |
|  |
|  |

* WebRISC-V:

|  |  |
| --- | --- |
| **WebRISC-V** | |
| Ventajas: |  |
| Como principal ventaja de este simulador, nos encontramos que la configuración e instalación necesarias son nulas, es decir, al ser un simulador web no es necesario que se realice instalación de ningún tipo, sino que se puede simplemente acceder a la página web. Por otro lado este simulador cuenta con un pequeño apartado cuando se está editando el código que muestra todas las posibles instrucciones y pseudoinstrucciones que se pueden usar, donde si pinchas en cualquiera da una pequeña descripción. Ya para finalizar, el apartado del pipeline es bastante gráfico, siendo que el simulador se centra principalmente en esto. | |
|  |
|  |
|  |
|  |  |  |
| Como desventaja principal pondría el entorno: al ser por página web, con muchos alumnos accediendo al mismo tiempo se podría sobresaturar, esto junto a otros problemas relacionados pueden volver esta una opción inviable. Por otro lado, no se puede deshabilitar la ejecución segmentada, donde la instrucción que se está ejecutando en el momento no está lo suficentemente clara, o puede llevar a confusión. También carece de apartado de cachés, y entrada salida. | |  |
|  |
|  |
|  |

* Eclipse RISCV:

|  |  |
| --- | --- |
| **Eclipse RISCV** | |
| Ventajas: |  |
| Como principal ventaja cabe destacar el uso de un entorno sobre el que los alumnos pueden estar relativamente familiarizados, en la creación de proyectos, debugging, etc. Por otro lado tenemos que se pueden compilar distintos archivos para un mismo programa, e incluso tener en un mismo proyecto ficheros ensamblador y ficheros c/c++, de la misma manera que trabajaríamos con el compilador por separado pero de forma más automatizada. | |
|  |
|  |
|  |
| Desventajas: |  |  |
| La principal desventaja de este entorno es su configuración, esta es relativamente compleja en la que hay que seguir varios pasos para poder armar un proyecto y empezar a trabajar con este. El apartado de configuración nos muestra opciones muy extensas que pueden ser abrumadoras para un usuario que recién esté tratando de empezar. Por otro lado este entorno no cuenta con un apartado gráfico ni para memoria caché, ni para pipeline. Y en cuanto a la entrada salida los periféricos se tendrían que añadir puesto que el propio apartado empieza de cero, es decir no cuenta ni siquiera con un ejemplo sobre el que apoyarse. | |  |
|  |
|  |
|  |

## 3.2 Selección Final

Finalmente, después de analizar los diferentes simuladores se seleccionaron dos posibles candidatos: *“RARS”* y *“RIPES”*. Inicialmente se ha visto que los simuladores de entorno web son demasiado problemáticos para usarlos como herramienta de trabajo. Por otra parte, simuladores como *“Venus”* y *“Eclipse RISCV”*, a pesar de contar con la ventaja de montarse sobre entornos en los que los estudiantes pueden estar más acostumbrados, no compensan en la interfaz e implementación. Este mismo defecto se puede encontrar también en el simulador *“Jupiter”* que es demasiado sencillo, ofreciendo solo un entorno con el que trabajar con el ensamblador RISC-V puro, con apenas alguna característica adicional a esto.

Por otro lado, para seleccionar un único simulador se han destacado las principales diferencias de estos últimos candidatos:

El simulador *“RIPES”* presenta una interfaz mucho más completa en configuración e implementación, los apartados son mucho más visibles y gráficos, se puede trabajar con código *“C”* el entorno es mucho más cómodo a la hora de depurar y ejecutar código, etc. Sin embargo, una de sus principales limitaciones es la implementación de la entrada salida, donde la única implementación de la misma es programada. En contraposición, el simulador “RARS”, aun cuando se mantiene por detrás en la mayoría de aspectos. Mantiene una implementación (aunque un tanto obsoleta) de los registros de control e instrucciones necesarias para trabajar con interrupciones y excepciones.

En este punto se decidió que a pesar de la limitación presentada en el simulador *“RIPES”* este todavía era el más apto para lo que se estaba buscando dentro del departamento. Planteando incluso suplir esta desventaja implementando en el propio simulador un sistema de interrupciones y excepciones.

# 4 Investigación Adicional

En la resolución y análisis de los diferentes simuladores, hubo algunos apartados adicionales sobre los que también fue necesario investigar para poder completar con una mayor profundidad esta tarea. Entre estos conceptos se encontraban principalmente: la entrada salida dentro del estándar RISC-V actual, y el trabajo con ficheros y códigos *“C”* junto a ficheros de código ensamblador RISC-V.

Estos puntos se verán en mayor profundidad dentro de los siguientes apartados.

## 4.1 Compiladores y Trabajo con varios ficheros de código

Continuar con el trabajo realizado con ripes y Eclipse para trabajar con varios ficheros de código y los compiladores disponibles dentro del estándar. Mencionar el trabajo mezclando código C con código ensamblador RISCV, junto a las experiencias del linker script y de como se puede usar esto con el simulador escogido (RIPES).

Ha completar

## 4.2 Entrada Salida

Explicación de la entrada salida brevemente y su definición dentro del estándar: registros de control, instrucciones especializadas (Zcsr), controladores de interrupciones para un mejor control de prioridad. Mención especial al simulador RARS y como implementa este las interrupciones y excepciones, también al intento de implementarlo sobre el simulador que se ha escogido junto a las limitaciones de RARS para defender esta decisión.

Ha completar

# 5 Conclusiones de la Beca

En este apartado se relatarán las conclusiones finales de la beca, junto a los resultados de aprendizaje de la misma.

Este apartado se dedicará a la exposición de las principales conclusiones de la beca. Desde los resultados de aprendizaje obtenidos a través de la investigación hasta las conclusiones personales que se han obtenido en la beca.

## 5.1 Resultados de Aprendizaje

Ganancias intelectuales de la beca: Formación en el estándar RISV dentro de sus funcionalidades principales y entrada salida, trabajo con ficheros de enlace, etc.

Ha completar

## 5.2 Conclusiones Personales

Crecimiento personal gracias a la beca: experiencia semi real, motivación, ayuda para proyectos de movilidad (Corea), experiencia adicional para un futuro TFG o trabajo (documentación), etc.

Ha completar

# 6 Anexos

## 6.1 Ripes

### 6.1.1 Simulador

#### 6.1.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], RIPES [6], sus posibilidades y sus principales funciones.

RIPES es un simulador del estándar RISC-V, compatible con varias implementaciones del mismo (rv32im, rv64im, rv32imc, etc.), de libre distribución, escrito principalmente en C++ y orientado a la ejecución sobre un único fichero ensamblador.

#### 6.1.1.2 Interfaz

RIPES cuenta con una implementación muy gráfica para la interfaz de trabajo desde la ejecución de los programas, la visualización de los espacios de memoria, las fuentes y periféricos, etc. Por otro lado, el trabajo general con el código es bastante cómodo, con una visualización muy gráfica y dinámica dentro del proceso de ejecución y depuración.

#### 6.1.1.3 Memorias Caché

Para la visualización de la memoria, como se había dicho con anterioridad, RIPES cuenta con una visualización muy gráfica y detallada. Se muestra bastante pulida, tanto para la memoria principal como para la caché, donde de distingue entre caché de datos y de instrucciones, y podemos ver exactamente donde se ubican los bloques junto a su contenido.

A diferencia de otros simuladores que solo muestran si se ha producido un fallo en caché, o muestran apenas un pequeño esbozo, esta implementación muestra todo lo contrario entrando en bastante detalle dentro del proceso.

**Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.**Figura 1.1

**Tabla

El contenido generado por IA puede ser incorrecto.**Figura 1.2

#### 6.1.1.4 Pipeline

Una de las características más distintivas de este simulador es sin lugar a dudas la implementación de la ejecución segmentada. Como se puede ver en la imagen 1.4 podemos ver todo el proceso de ejecución por cada etapa, que además es adaptable a la cantidad de etapas que establezcamos dentro de lo implementado. Cuando ejecutamos un programa podemos ver también las instrucciones que se están ejecutando en este momento resaltadas en rojo que se van actualizando a cada paso que se da en el proceso de ejecución.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto. Figura 1.3

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto. Figura 1.4

#### 6.1.1.5 Entrada Salida

Incluso cuando este simulador cuenta con un apartado específico para la entrada salida, la implementación de la misma es solo programada puesto que las interrupciones y excepciones no están implementadas en el simulador. Adicionalmente a esto, solo cuenta con tres periféricos implementados.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto. Figura 1.5

#### 6.1.1.6 Trabajo con el simulador

Para trabajar con un único fichero, en el que se encuentre contenido todo el código del programa este simulador es casi ideal: cuenta con un editor en el que podemos ver a su misma vez el código fuente junto al desensamblado cuando se carga el programa ejecutable, cuenta con un sistema de búsqueda por etiquetas en el programa, se resalta la instrucción que se está ejecutando en el momento y cuenta con un sistema de ejecución automática con una velocidad configurable dentro de unos límites.

Sin embargo, si contamos con un proyecto formado por varios ficheros de código, o incluso si queremos ver varios proyectos en el simulador y trabajar con ellos de una manera cómoda, como si de un editor de texto general se tratase, nos encontramos en un aprieto puesto que este simulador no cuenta con soporte para la visualización de varios ficheros al mismo tiempo, y adicionalmente, en caso de querer generar un programa a partir de varios ficheros de código se tendrá que montar desde fuera puesto que no se cuenta con soporte esto desde el propio simulador.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto. Figura 1.6

### 6.1.2 Instalación y Ejecución

Dentro de este apartado se describirá el proceso necesario para la instalación y configuración inicial del simulador tanto para los sistemas operativos tipo Windows como tipo Linux.

#### 6.1.2.1 Linux

Dentro del apartado “Releases” del repositorio oficial [6] nos encontramos con las diferentes versiones que hay del simulador hasta la fecha. En este punto se elige preferentemente la versión más actualizada y se procede con la selección de la versión adaptada para el sistema operativo sobre el que se vaya a trabajar, en este caso, Linux.Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto. Figura 2.1

Una vez se tiene instalado y ubicado el archivo de extensión “AppImage” en el sistema, ya se podrá iniciar el simulador ejecutando este mismo archivo. Es posible que la ejecución falle porque no se tienen instaladas las dependencias necesarias en el sistema.

Texto

El contenido generado por IA puede ser incorrecto. Figura 2.2

En este caso solo se necesitaría instalar las mismas para poder empezar a trabajar, en este caso no se cuenta con las librerías de fuse [7]. Si se cuenta con apt en el sistema, se puede hacer con la siguiente sentencia: “sudo apt install libfuse2”.

#### 6.1.2.2 Windows

Dentro del apartado “Releases” del repositorio oficial [6] nos encontramos con las diferentes versiones que hay del simulador hasta la fecha. En este punto se elige preferentemente la versión más actualizada y se procede con la selección de la versión adaptada para el sistema operativo sobre el que se vaya a trabajar, en este caso Windows.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.** Figura 2.3

Una vez hemos instalado el archivo comprimido, archivo de extensión zip. Para iniciar el simulador se tendrán que extraer los archivos comprimidos y llamar al ejecutable dentro de estos: “ripes.exe”.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto. Figura 2.4

### 6.1.3 Aspectos Destacables y Limitaciones

Adicionalmente, como aspecto positivo a resaltar de este simulador frente a otros, RIPES permite cargar programas ya generados desde fuera del simulador, es decir: si se cuenta con un programa ejecutable ya generado el simulador puede procesarlo como un programa normal, aunque no se podrá modificar el código desde el mismo. Esto le permite suplir la incapacidad de generar o trabajar con proyectos compuestos por varios ficheros de código, puesto que, aunque no se puedan generar desde dentro, se pueden generar fuera y seguir ejecutándose desde dentro.

Siguiendo con el anterior punto, si se cuenta con el compilador de C para RISC-V adecuado, se puede trabajar con programas que estén escritos en C, ya no solo en ensamblador. Y si se genera el programa desde fuera, podrían estar escritos en otros lenguajes de programación como por ejemplo C++, o inclusive mezclar varios ficheros de código escritos en distintos lenguajes de programación como C y ensamblador.

A pesar de que el propio simulador es uno de los más completos dentro de su entorno, podemos encontrar algunos aspectos a mejorar. Por ejemplo: no cuenta con una implementación de interrupciones ni excepciones lo que puede dificultar la depuración de los errores; por otro lado, no cuenta con una instrucción para para la finalización del programa, y cuando finaliza no se puede volver una instrucción hacia atrás como cuando se ejecuta normalmente, sino que se tiene que reiniciar la ejecución del programa desde el principio. Para finalizar, tenemos que el trabajo con varios ficheros de código, o incluso con varios proyectos al mismo tiempo se vuelve bastante incómodo siendo que no puede trabajar con el mismo simulador como se podría con otros del estilo como RARS por ejemplo.

Como punto por fuera del simulador, el lenguaje y formato en que está escrito este simulador se puede considerar bastante complejo siendo casi una simulación hardware, lo que complica el añadir características o configuraciones a medida, o incluso cambiar algunas de las ya existentes.

## 6.2 RARS

### 6.2.1 Simulador

#### 6.2.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], RARS [8], sus posibilidades y sus principales funciones.

RARS es un simulador de libre distribución que permite la ejecución de código ensamblador RISC-V. Este simulador cuenta con una interfaz muy gráfica para algunos aspectos, donde se especializa en el tratamiento de programas constituidos por un único fichero ensamblador RISC-V.

#### 6.2.1.2 Interfaz

La interfaz de RARS, como se había mencionado con anterioridad es una interfaz gráfica que nos permite visualizar el flujo del programa resaltando las instrucciones que se están ejecutando en el momento.

#### 6.2.1.3 Memorias Caché

Para la visualización de la memoria, dentro del apartado de ejecución podemos ver en la sección inferior el contenido de las direcciones de memoria que especifiquemos, e inclusive cambiar algunos de los valores. También podemos filtrar algunas de las secciones específicas del programa como el código, los datos o la pila; y movernos a partir de estas direcciones. O de otra forma, especificar una dirección en concreto y visualizar sus valores o como se había mencionado antes, modificarlos con libertad.**Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.**Figura 1.1

Por otro lado, aunque RARS cuenta con una implementación de memorias caché, esta es muy pobre: su representación no muestra el contenido de la misma, sino que solo marca con color si se ha producido un fallo en caché, y en caso de ampliar la memoria caché lo suficiente los colores dejan de apreciarse o incluso desaparecen. Esto sin contar que hay que conectar esta memoria al programa de forma que cada vez que se produzca un parón en el programa o se reinicie el mismo, habrá que reiniciar y reconectar de nuevo la memoria caché.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Figura 1.2

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.Figura 1.3

#### 6.2.1.4 Pipeline

Este simulador no cuenta con una implementación de la ejecución segmentada, sino que solo ejecuta instrucción a instrucción.

#### 6.2.1.5 Entrada Salida

Dentro de todo lo implementado en este simulador, sin duda alguna una de las implementaciones por los que más se destaca es la entrada salida, que cuenta con la implementación de excepciones e interrupciones junto a una serie de registros de control a nivel de usuario. Esto, conforme al estándar del momento en el que se implementó, que en algunos aspectos se ha quedado obsoleto, por ejemplo: en la gestión de interrupciones y excepciones a nivel de usuario.

Sin embargo, incluso cuando cuenta con algunos periféricos implementados con los que se puede jugar con la entrada salida y las interrupciones, las instrucciones sobre como estos funcionan no están muy claras, y adicionalmente tienen algunas fallas y aspectos a mejorar en el funcionamiento de los mismos.

#### 6.2.1.6 Trabajo con el simulador

Para trabajar con el simulador, contamos con una interfaz con la que se puede editar varios archivos al mismo tiempo, como si de un editor de texto general se tratase, sin embargo, solo permite el cargado de programas montados sobre un único fichero de código ensamblador RISC-V.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.Figura 1.5

Por otro lado, cuando se carga el programa dejamos de ver el código original y pasamos a ver solo la interfaz de ejecución con el código ya generado. Sin embargo, esta interfaz es bastante detallada y práctica para la depuración y ejecución del código.

Tabla

El contenido generado por IA puede ser incorrecto.Figura 1.6

### 6.2.2 Instalación y Ejecución

En el proceso de instalación y configuración inicial de este simulador no se puede notar ninguna diferencia significativa entre distintos sistemas operativos, siendo, la única dependencia necesaria para que este simulador funcione: tener instalada una versión del JDK de Java 8 o superior.

Entonces, para instalar RARS, nos referimos al apartado releases del repositorio oficial de RARS [9] donde descargamos el archivo de extensión jar.Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto. Figura 2.1

Una vez instalado el ejecutable, para ejecutar el simulador tenemos que ejecutar el archivo de extensión jar con la sentencia “java -jar”, y si todo se ha instalado correctamente, entonces se debería abrir la interfaz del simulador.

### 6.2.3 Aspectos Destacables y Limitaciones

Adicionalmente, un aspecto positivo con el que cuenta este simulador es la simplicidad del mismo en su implementación, es decir, que está escrito en un lenguaje amigable como es java y de una forma fácil de entender y con la que es fácil añadir funcionalidades.

A pesar de que el propio simulador cuenta con bastantes ventajas, podemos encontrar una serie de fallas en la implementación: varios bugs visuales en algunos de los apartados e incluso algunos errores en el funcionamiento de los periféricos. Pero por sobre esto, probablemente lo más significativo sea la desactualización de la implementación, cuya última actualización data de inicios del 2020, donde podemos ver algunos puntos, como la entrada salida, que se han quedado desfasados con el estándar actual.

## 6.3 Jupiter

### 6.3.1 Simulador

#### 6.3.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1] Jupiter [10], simulador de entorno gráfico a modo de aplicación.

Este es un simulador relativamente simple dentro de su entorno, muy especializado en la ejecución de programas de ensamblador puro.

#### 6.3.1.2 Interfaz

Jupiter es un simulador muy sencillo pensado para trabajar solo con un fichero ensamblador RISC-V, donde las funciones y los apartados de configuración son relativamente sencillos en su implementación. Sin contar que apenas tiene implementadas las operaciones básicas para números enteros con multiplicación y división.

Vemos entonces una interfaz sobria con lo mínimo e imprescindible para trabajar con el, esto puede constituir tanto una ventaja como una desventaja: aunque no aturulla al programador con un exceso de información, ciertas características si se echan de menos cuando se está ejecutando y depurando código.

#### 6.3.1.3 Memorias Caché

Este simulador cuenta con un pequeño apartado para la visualización del contenido en memoria, y con una implementación sencilla de las memoras caché por defecto, con algunas opciones de configuración. Sin embargo, esta puede dejar bastante que desear siendo que al igual que simuladores como RARS, no muestra el contenido sino solo si se ha producido un fallo en caché, y al aumentar el tamaño se vuelve mucho más difícil de manejar. Por otro lado, la visualización de la memoria principal no ofrece nada destacable por sobre otros simuladores solo se tienen un pequeño apartado para el trabajo con la misma.

Calendario

El contenido generado por IA puede ser incorrecto.Figura 1.1 (Visualización del contenido en memoria en Jupiter)

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Figura 1.2 (Apartado de memoria caché de Jupiter)

#### 6.3.1.4 Pipeline

Este simulador no cuenta con una implementación de la ejecución segmentada, sino que solo ejecuta instrucción a instrucción.

#### 6.3.1.5 Entrada Salida

Este simulador, al igual que muchos de su entorno, no cuenta con una implementación de excepciones ni interrupciones, en este caso no teniendo ni siquiera implementados periféricos con los que trabajar la entrada salida, incluso si esta es programada.

#### 6.3.1.6 Trabajo con el simulador

Trabajar con este simulador se puede hacer algo incómodo en comparación a otros simuladores: cuando se carga el programa el código no es visible, por otra parte, solo se pueden visualizar o los registros o la memoria. Sin embargo, la sencillez que aporta el simulador en cuenta al procesamiento del programa hace que sea muy fácil acostumbrarse a el por lo que tiene ese punto a favor.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Figura 1.3

### 6.3.2 Instalación y Ejecución

En el proceso de instalación y configuración inicial de este simulador hay que referirse al apartado installation del repositorio oficial [10] donde tendremos que instalar el archivo comprimido pertinente al sistema operativo que tengamos disponible.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**Figura 2.1

Una vez extraídos todos los archivos del fichero comprimido, para iniciar el simulador solo tendremos que llamar al ejecutable con su mismo nombre (jupiter) que se encuentra en la carpeta “/image/bin”. Para poder ejecutar el programa el único prerrequisito con el que contamos es tener instalada una versión del JDK de java 8 o superior, de manera contraria la ejecución fallará.

### 6.3.3 Aspectos Destacables y Limitaciones

Adicionalmente, un aspecto positivo con el que cuenta este simulador es la simplicidad del mismo en su implementación, es decir, que está escrito en un lenguaje amigable como es java y de una forma fácil de entender y con la que es fácil añadir funcionalidades. Cuenta con toda so documentación disponible [11].

Este simulador no ha presentado mayor problemática ni en su instalación, ni en su configuración ni en su ejecución. Quizá pudiendo atribuírselo a la sencillez del mismo simulador, siendo quizá esta la mayor de sus desventajas. Mientras que otros simuladores tienen implementaciones vastas y detalladas, este cuenta con lo mínimo e imprescindible para trabajar con el ensamblador, dando lugar a un entorno sumamente específico.

## 6.4 RISC-V Venus Simulator

### 6.4.1 Simulador

#### 6.4.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1]: Venus Simulator [12]. Este simulador es un tanto especial puesto que se monta sobre uno de los entornos de programación más usados del sector, Visual Studio Code, aplicación de Microsoft disponible tanto para sistemas operativos Windows como para sistemas operativos Linux, como una extensión de libre desarrollo.

#### 6.4.1.2 Interfaz

La interfaz del simulador es la misma que la del entorno, siendo que la única diferencia con respecto al editor de texto, es cuando se abre la interfaz para el trabajo de depuración, donde se abren algunos apartados adicionales: para visualización de los registros, memoria, campos, etc.

#### 6.4.1.3 Memorias Caché

Este simulador cuenta con un pequeño apartado para la visualización del contenido de la memoria, sin embargo, no cuenta con un apartado de memorias caché.

#### 6.4.1.4 Pipeline

Este simulador cuenta con una única implementación que ejecuta en un único nivel, es decir, no cuenta con opciones para ejecución segmentada.

#### 6.4.1.5 Entrada Salida

Este simulador cuenta con algunos periféricos implementados con los que se puede trabajar con la entrada salida programada. Sin embargo, no cuenta con una implementación de interrupciones ni excepciones.

#### 6.4.1.6 Trabajo con el simulador

Para trabajar con el simulador se puede trabajar muy cómodamente teniendo algunos problemillas de entorno.

### 6.4.2 Instalación y Ejecución

Este simulador se instala como extensión de la aplicación de Microsoft: Visual Studio Code. Es por esto que el proceso de instalación es prácticamente inmediato siempre que se cuente con la aplicación ya instalada. Aplicación que está disponible independientemente del sistema operativo [13].

Para ejecutar el simulador tenemos que trabajar con un fichero ensamblador de extensión “.s” donde tenemos que referirnos al botón de ejecución. Una vez hecho esto podremos ver el inicio de la interfaz y tendremos disponibles los registros y variables del mismo.

### 6.4.3 Aspectos Destacables y Limitaciones

Como aspecto a destacar, tenemos que este simulador se desarrolla sobre uno de los entornos más usados dentro del mundo de la programación, siendo que la familiaridad con el entorno puede ayudar al programador en sí mismo.

Por otro lado, este sistema tiene algunas limitaciones, entre ellas el formato y visualización de los registros, que se ven como variables, teniendo muy poco espacio de interfaz dedicado. También cabe mencionar que no cuenta con ninguna implementación interesante como puede ser la entrada salida por interrupciones, un apartado para memorias caché, ejecución segmentada, etc. Esto, junto a la escasa configuración posible del simulador, lo que lo vuelve altamente específico para programar con un único fichero ensamblador.

## 6.5 EmulsiV

### 6.5.1 Simulador

#### 6.5.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], EmulsiV [14] [15], simulador web de código abierto escrito principalmente en JScript.

Este simulador es bastante sencillo, más pensando como ejemplo académico que como herramienta para probar código, siendo que viene con una serie de ejemplos precargados que son los únicos programas que se pueden cargar en un principio.

#### 6.5.1.2 Interfaz

Este simulador solo cuenta con una interfaz, conteniendo esta todos los apartados y puntos del simulador, para trabajar con todos y cada uno de los ejemplos solo se cuenta con esta única página. Sin embargo algunos de los apartados de la misma cambian a lo largo de la ejecución: como el apartado de ejecución segmentada que a medida que se ejecuta el código este cambia siguiendo la misma ejecución.

#### 6.5.1.3 Memorias Caché

Aunque este simulador cuenta con un pequeño apartado para la visualización del contenido de la memoria, no cuenta con una implementación de memorias caché.

Tabla

El contenido generado por IA puede ser incorrecto.Figura 1.1 (Apartado para el contenido de la memoria principal)

#### 6.5.1.4 Pipeline

Este simulador cuenta con una implementación de ejecución segmentada, este apartado es muy gráfico en el que se sigue la ejecución paso a paso. Sin embargo, este apartado es el corazón del simulador y por tanto no se puede ni configurar ni desactivar.

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.Figura 1.2

#### 6.5.1.5 Entrada Salida

Como aspecto interesante, en este simulador se ha implementado un sistema de entrada y salida por interrupciones con unos periféricos definidos. Sin embargo, se trabaja de una manera transparente y no se muestran ni el proceso ni los registros de control involucrados, si no que simplemente se salta al tratamiento de la interrupción.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Figura 1.3

#### 6.5.1.6 Trabajo con el simulador

El trabajo con este simulador es plenamente teórico, consistiendo en las pruebas de las que han dispuesto los desarrolladores, casi como herramienta de enseñanza con una serie de ejemplos precompilados para introducir el concepto. No como una herramienta con la que jugar con libertad e intentar explotar los límites del estándar. También, relacionado a esto, no parece preparado para la depuración y edición de código.

### 6.5.2 Instalación y Ejecución

Este simulador está disponible en el enlace oficial, por lo que no sería necesario realizar ningún proceso de instalación ni configuración previos.

Por otro lado, podremos recoger el código fuente y subirlo sobre un servidor para trabajar sin conexión con estos sencillos pasos. Primero clonamos el repositorio (enlace oficial en la sección de Documentación), y ejecutamos la sentencia “npm install” dentro del directorio principal que acabamos de clonar, para iniciar la configuración del servidor local que vamos a usar.

Texto

Descripción generada automáticamenteFigura 2.1

Una vez configurado el servidor lo único que tenemos que hacer es arrancarlo para poder acceder al simulador. Esto se hace ejecutando la sentencia “npm start”.

Texto

Descripción generada automáticamenteFigura 2.2

Una vez hecho esto si todo ha ido correctamente, usando una de las direcciones que nos devuelve podremos conectarnos al servidor local que hemos lanzado donde se encontrara el simulador ya preparado.

### 6.5.3 Aspectos Destacables y Limitaciones

Al tratarse de un simulador que se ejecuta sobre un entorno web podríamos decir tanto que tiene ventajas como desventajas, por ejemplo: por una parte, no existe una instalación para el usuario, sin embargo, el proveedor tendrá que dar soporte a la cantidad de usuarios que se presenten, lo que puede ser problemático dependiendo de la cantidad de usuarios que interactúen al mismo tiempo.

Por otro lado, el mayor problema que presenta este simulador es la limitación del mismo: no se puede editar el código, ni trabajar con otro código que no nazca de los ejemplos, y luego, aunque cuenta con varios puntos interesantes como pueden ser las interrupciones y la ejecución segmentada. Cuenta con una escasa posibilidad de configuración sobre las mismas y sobre el entorno en general.

## 6.6 Creator

### 6.6.1 Simulador

#### 6.6.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], Creator [16] [17] [18], simulador web pensado especialmente para trabajar con un único fichero ensamblador.

#### 6.6.1.2 Interfaz

La interfaz del simulador es una interfaz simple, donde contamos con un apartado específico para modificar y escribir código. Y con el apartado general donde se ejecutará y depurará: un apartado donde se pueden ver los registros, el contenido de memoria, la instrucción que se está ejecutando, añadir puntos de ruptura, etc.

Esta interfaz es relativamente amigable en el sentido de que no sobrecarga con opciones al usuario si no que se encuentra todo relativamente bien organizado.

#### 6.6.1.3 Memorias Caché

Aunque este simulador tiene un pequeño apartado para la visualización del contenido de las variables en memoria, no cuenta con una memoria caché implementada. Por otro lado, el contenido de memoria se va actualizando para ver a que está apuntando cada variable.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.Figura 1.1

#### 6.6.1.4 Pipeline

Este simulador no cuenta con una implementación de la ejecución segmentada, sino que solo ejecuta instrucción a instrucción.

#### 6.6.1.5 Entrada Salida

Este simulador no cuenta con implementación ninguna de entrada salida, ni por interrupciones, ni programada.

#### 6.6.1.6 Trabajo con el simulador

El trabajo con la ejecución y depuración del código es relativamente amigable, a medida que vas paso a paso puedes ver las actualizaciones resaltadas en los valores de memoria y las variables. También puedes ver a donde apuntan los punteros de los registros, no hay un exceso de información por lo que tampoco abruma como otros simuladores del entorno.

### 6.6.2 Instalación y Ejecución

Este simulador se ejecuta sobre un entorno web, donde no se requiere configuración y dependencia previa más que contar con un navegador por el que se pueda acceder a la página del mismo.

### 6.6.3 Aspectos Destacables y Limitaciones

Como aspecto a destacar, el trabajo con el simulador dentro de un entorno de un único fichero ensamblador es relativamente cómodo, con varias características únicas que ayudan en la depuración.

Por otro lado, al tratarse de un simulador web, generar problemas de estabilidad y dependencia de la red, también lo podemos ver bastante simple en cuanto al desarrollo de aspectos que pueden resultar interesantes como: entrada salida por interrupciones, memorias caché, ejecución segmentada, etc.

## 6.7 WebRISC-V

### 6.7.1 Simulador

#### 6.7.1.1 Introducción

Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], WebRISC-V [19] [20], simulador web de libre desarrollo y distribución.

#### 6.7.1.2 Interfaz

La interfaz de este simulador es única, donde se reserva una sección para la edición de código. Sin embargo, con la gran cantidad de elementos que se presentan en la página, y la distribución de los mismos, cuando se está depurando el código, se vuelve una tarea compleja entender el flujo general de la ejecución.

Por otro lado, el simulador presenta la ejecución de una manera gráfica, con elementos dinámicos en la ejecución segmentada.

#### 6.7.1.3 Memorias Caché

Aunque este simulador cuenta con un pequeño apartado para la visualización de la memoria, no cuenta con una implementación de memorias caché. Tabla

El contenido generado por IA puede ser incorrecto.Figura 1.1

#### 6.7.1.4 Pipeline

Uno de los apartados principales de este simulador es la ejecución segmentada, que se muestra de una manera gráfica. Esta implementación es muy completa, sin embargo, precisamente por esto puede abrumar al usuario en su implementación.

Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.Figura 1.2

#### 6.7.1.5 Entrada Salida

Este simulador no cuenta con ningún tipo de implementación para la entrada salida: ni periféricos, ni interrupciones, ni excepciones.

#### 6.7.1.6 Trabajo con el simulador

A la hora de trabajar con este simulador nos encontramos con una interfaz un tanto incómoda, aunque con algunas ventajas. Por una parte, podemos visualizar todas las instrucciones implementadas junto a la traducción de todas las pseudoinstrucciones. Pero por la otra tenemos una interfaz para editar el código que se queda pequeña en muchos casos.

En la depuración y ejecución es difícil seguir el hilo de las instrucciones que se están ejecutando en ese momento.

### 6.7.2 Instalación y Ejecución

Con este simulador, al tratarse de un simulador web, no hay un proceso de instalación en si mismo, si no que se dispone de un enlace sobre el que se puede acceder al simulador mediante un navegador general.

Por otro lado, si se busca tener una copia local para poder ejecutar el mismo sin conexión, se puede preparar por Docker o con un servidor apache. Para más información sobre este proceso y todos sus pasos visitar el enlace al proceso de instalación [21].

### 6.7.3 Aspectos Destacables y Limitaciones

Como aspectos a destacar de este simulador podemos sacar algunas conclusiones. Primero podemos ver un entorno muy completo para la ejecución segmentada.

Por otra parte, este es un simulador muy limitado, con muy pocas opciones de configuración y un entorno que solo permite la ejecución de proyectos conformados por un único fichero ensamblador. Añadiendo a esto, el entorno de depuración se queda muy por detrás. Añadiendo a esto que la interfaz de depuración y ejecución deja mucho que desear.

# 7 Referencias

|  |  |
| --- | --- |
| [1] | RISC-V, «Ratified RISC-V Specifications,» [En línea]. Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications. |
| [2] | Tomshardware, «Nvidia to ship a billion of RISC-V cores in 2024,» [En línea]. Available: https://www.tomshardware.com/pc-components/gpus/nvidia-to-ship-a-billion-of-risc-v-cores-in-2024. |
| [3] | SiFIve, «The History of RISC‑V,» [En línea]. Available: https://www.sifive.com/about/risc-v-history. |
| [4] | SiFive, «SiFive Performance P800-Series,» [En línea]. Available: https://www.sifive.com/cores/performance-p870d. |
| [5] | riscv-software-src, «riscv-isa-sim,» [En línea]. Available: https://github.com/riscv-software-src/riscv-isa-sim. |
| [6] | mortbopet, «Repositorio Oficial de RIPES,» [En línea]. Available: https://github.com/mortbopet/Ripes. |
| [7] | Fuse, «FUSE,» [En línea]. Available: https://github.com/AppImage/AppImageKit/wiki/FUSE. |
| [8] | TheThirdOne, «Rars Repository,» [En línea]. Available: https://github.com/TheThirdOne/rars. |
| [9] | TheThirdOne, «RARS Releases,» [En línea]. Available: https://github.com/TheThirdOne/rars/releases/tag/v1.6. |
| [10] | andrescv, «/andrescv/jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter. |
| [11] | «Jupiter,» [En línea]. Available: https://jupitersim.gitbook.io/jupiter/es. |
| [12] | hm-riscv, «/hm-riscv/vscode-riscv-venus,» [En línea]. Available: https://github.com/hm-riscv/vscode-riscv-venus. |
| [13] | «Visual Studio Marketplace RISC-V Venus Simulator,» [En línea]. Available: https://marketplace.visualstudio.com/items?itemName=hm.riscv-venus. |
| [14] | ESEO-Tech, «/ESEO-Tech/emulsiV,» [En línea]. Available: https://github.com/ESEO-Tech/emulsiV. |
| [15] | ESEO-Tech, «/ESEO-Tech/emulsiV/doc,» [En línea]. Available: https://eseo-tech.github.io/emulsiV/doc/. |
| [16] | [En línea]. Available: https://creatorsim.github.io/creator/. |
| [17] | [En línea]. Available: https://github.com/creatorsim/creator. |
| [18] | [En línea]. Available: https://creatorsim.github.io/creator/. |
| [19] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V. |
| [20] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki. |
| [21] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation. |
| [22] | mortbopet, «/Ripes/Docs,» [En línea]. Available: https://github.com/mortbopet/Ripes/tree/master/docs. |
| [23] | andrescv, «Repositorio Oficial Jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter. |
| [24] | «Guía Instalación WebRISC-V,» [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation. |