

**Informe Final: Beca de Colaboración**



Autor: Bruno burgos kosmalski

Índice

[1 Introducción 4](#_Toc198729921)

[2 Primeros Pasos 4](#_Toc198729922)

[2.1 ¿Qué es RISC-V? 4](#_Toc198729923)

[2.2 ¿Cómo comenzó la investigación? 6](#_Toc198729924)

[3 Simuladores 7](#_Toc198729925)

[3.1 Selección Inicial 7](#_Toc198729926)

[3.2 Selección Final 13](#_Toc198729927)

[4 Investigación Adicional 14](#_Toc198729928)

[4.1 Compiladores y Trabajo con varios ficheros de código 14](#_Toc198729929)

[4.2 Entrada Salida 14](#_Toc198729930)

[5 Conclusiones de la Beca 15](#_Toc198729931)

[5.1 Resultados de Aprendizaje 15](#_Toc198729932)

[5.2 Conclusiones Personales 15](#_Toc198729933)

[6 Anexos 16](#_Toc198729934)

[6.1 Ripes 16](#_Toc198729935)

[6.1.1 Simulador 16](#_Toc198729936)

[6.1.1.1 Introducción 16](#_Toc198729937)

[6.1.1.2 Interfaz 16](#_Toc198729938)

[6.1.1.3 Memorias Caché 16](#_Toc198729939)

[6.1.1.4 Pipeline 17](#_Toc198729940)

[6.1.1.5 Entrada Salida 18](#_Toc198729941)

[6.1.1.6 Trabajo con el simulador 19](#_Toc198729942)

[6.1.2 Instalación y Ejecución 20](#_Toc198729943)

[6.1.2.1 Linux 20](#_Toc198729944)

[6.1.2.2 Windows 21](#_Toc198729945)

[6.1.3 Aspectos Destacables y Limitaciones 22](#_Toc198729946)

[6.2 RARS 22](#_Toc198729947)

[6.2.1 Simulador 23](#_Toc198729948)

[6.2.1.1 Introducción 23](#_Toc198729949)

[6.2.1.2 Interfaz 23](#_Toc198729950)

[6.2.1.3 Memorias Caché 23](#_Toc198729951)

[6.2.1.4 Pipeline 24](#_Toc198729952)

[6.2.1.5 Entrada Salida 24](#_Toc198729953)

[6.2.1.6 Trabajo con el simulador 25](#_Toc198729954)

[6.2.2 Instalación y Ejecución 25](#_Toc198729955)

[6.2.3 Aspectos Destacables y Limitaciones 26](#_Toc198729956)

[6.3 Jupiter 26](#_Toc198729957)

[6.3.1 Simulador 26](#_Toc198729958)

[6.3.1.1 Introducción 26](#_Toc198729959)

[6.3.1.2 Interfaz 26](#_Toc198729960)

[6.3.1.3 Memorias Caché 27](#_Toc198729961)

[6.3.1.4 Pipeline 27](#_Toc198729962)

[6.3.1.5 Entrada Salida 28](#_Toc198729963)

[6.3.1.6 Trabajo con el simulador 28](#_Toc198729964)

[6.3.2 Instalación y Ejecución 29](#_Toc198729965)

[6.3.3 Aspectos Destacables y Limitaciones 29](#_Toc198729966)

[6.4 RISC-V Venus Simulator 29](#_Toc198729967)

[6.4.1 Simulador 30](#_Toc198729968)

[6.4.1.1 Introducción 30](#_Toc198729969)

[6.4.1.2 Interfaz 30](#_Toc198729970)

[6.4.1.3 Memorias Caché 30](#_Toc198729971)

[6.4.1.4 Pipeline 30](#_Toc198729972)

[6.4.1.5 Entrada Salida 30](#_Toc198729973)

[6.4.1.6 Trabajo con el simulador 30](#_Toc198729974)

[6.4.2 Instalación y Ejecución 30](#_Toc198729975)

[6.4.3 Aspectos Destacables y Limitaciones 31](#_Toc198729976)

[6.5 EmulsiV 31](#_Toc198729977)

[6.5.1 Simulador 31](#_Toc198729978)

[6.5.1.1 Introducción 31](#_Toc198729979)

[6.5.1.2 Interfaz 31](#_Toc198729980)

[6.5.1.3 Memorias Caché 31](#_Toc198729981)

[6.5.1.4 Pipeline 32](#_Toc198729982)

[6.5.1.5 Entrada Salida 32](#_Toc198729983)

[6.5.1.6 Trabajo con el simulador 33](#_Toc198729984)

[6.5.2 Instalación y Ejecución 33](#_Toc198729985)

[6.5.3 Aspectos Destacables y Limitaciones 34](#_Toc198729986)

[6.6 Creator 34](#_Toc198729987)

[6.6.1 Simulador 34](#_Toc198729988)

[6.6.1.1 Introducción 34](#_Toc198729989)

[6.6.1.2 Interfaz 34](#_Toc198729990)

[6.6.1.3 Memorias Caché 35](#_Toc198729991)

[6.6.1.4 Pipeline 35](#_Toc198729992)

[6.6.1.5 Entrada Salida 35](#_Toc198729993)

[6.6.1.6 Trabajo con el simulador 35](#_Toc198729994)

[6.6.2 Instalación y Ejecución 35](#_Toc198729995)

[6.6.3 Aspectos Destacables y Limitaciones 35](#_Toc198729996)

[6.7 WebRISC-V 36](#_Toc198729997)

[6.7.1 Simulador 36](#_Toc198729998)

[6.7.1.1 Introducción 36](#_Toc198729999)

[6.7.1.2 Interfaz 36](#_Toc198730000)

[6.7.1.3 Memorias Caché 36](#_Toc198730001)

[6.7.1.4 Pipeline 36](#_Toc198730002)

[6.7.1.5 Entrada Salida 37](#_Toc198730003)

[6.7.1.6 Trabajo con el simulador 37](#_Toc198730004)

[6.7.2 Instalación y Ejecución 37](#_Toc198730005)

[6.7.3 Aspectos Destacables y Limitaciones 37](#_Toc198730006)

[6.9 Trabajo con “C” y ensamblador RISC-V con RIPES 38](#_Toc198730007)

[6.9.1 Introducción 38](#_Toc198730008)

[6.9.2 Configuración y requisitos previos 38](#_Toc198730009)

[6.9.2.1 Pasos a seguir 38](#_Toc198730010)

[6.9.3 Trabajar con la Interfaz de RIPES 39](#_Toc198730011)

[6.9.4 Trabajar a partir de Ejecutables 43](#_Toc198730012)

[6.9.5 Usando fichero de enlace 48](#_Toc198730013)

[6.9.6 Resumen 51](#_Toc198730014)

[6.10 Privilegios e Interrupciones en RISC-V 52](#_Toc198730015)

[6.10.1 Aclaración Inicial 52](#_Toc198730016)

[6.10.2 Introducción 52](#_Toc198730017)

[6.10.3 Niveles de Privilegios 53](#_Toc198730018)

[6.10.4 Registros de Control y Estado (CSR): 54](#_Toc198730019)

[6.10.5 Instrucciones privilegiadas adicionales 59](#_Toc198730020)

[6.10.6 Resumen 59](#_Toc198730021)

[6.10.7 Información Adicional (Controladores) 60](#_Toc198730022)

[6.10.8 Listado de Algunos de los Registros de Control y Estado 61](#_Toc198730023)

[6.10.9 Terminología Usada 64](#_Toc198730024)

[6.11 Entrada Salida mediante interrupciones con RARS 64](#_Toc198730025)

[6.11.1 Introducción 64](#_Toc198730026)

[6.11.2 Registros de Control e Instrucciones útiles 64](#_Toc198730027)

[6.11.2.1 Registros 65](#_Toc198730028)

[6.11.2.2 Instrucciones 65](#_Toc198730029)

[6.11.3 Excepciones 66](#_Toc198730030)

[6.11.4 Entrada/Salida por Interrupciones 67](#_Toc198730031)

[6.11.5 Resumen 73](#_Toc198730032)

[6.12 PLIC 73](#_Toc198730033)

[6.12.1 Introducción 73](#_Toc198730034)

[6.12.2 Flujo de las Interrupciones 74](#_Toc198730035)

[6.12.3 Componentes y Adicionales 75](#_Toc198730036)

[6.12.3.1 Interrupt Priority regs 75](#_Toc198730037)

[6.12.3.2 Interrupt Pending bits 75](#_Toc198730038)

[6.12.3.4 Interrupt Enable regs 75](#_Toc198730039)

[6.12.3.5 Priority Thresholds 75](#_Toc198730040)

[6.12.3.6 Interrupt Claim and Completion register 75](#_Toc198730041)

[6.12.4 Conclusión 76](#_Toc198730042)

[7 Referencias 77](#_Toc198730043)

# 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”* [6]*.*

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

En un punto de la beca se propuso el trabajo con ficheros de código “C” y ensamblador RISC-V, puesto que dos de los simuladores propuestos podían trabajar con ello: RIPES y Eclipse RISCV.

Para esto se estuvo mirando los compiladores del estándar disponibles y como trabajar con ellos, se miró la investigación con los ficheros de enlace para controlar la creación de los programas.

Esto lo podemos ver en el [anexo 9](#_6.9_Trabajo_con), dedicado al trabajo con “C” y ensamblador RISC-V trabajando con el simulador escogido para el desarrollo del mismo, RIPES.

## 4.2 Entrada Salida

Una de las diferencias clave en la implementación del simulador RARS con respecto a RIPES es el apartado dedicado a la entrada salida. En el simulador RARS se explora una implementación más detallada de las interrupciones y excepciones con respecto al estándar: contamos con las instrucciones necesarias (Extensión Zicsr), con los registros de control y algunos periféricos con los que poder probar este punto.

Para poder encajar todo este concepto se estuvieron investigando los fundamentos del estándar, desde los registros fundamentales y su comportamiento, pasando por las instrucciones hasta los controladores del estándar, puesto que el mismo deja bastante que desear en concepto de control de las interrupciones externas por sí mismo.

Estas partes las podemos ver en los anexos 10, 11 y 12 de forma mucho más detallada. En el [anexo 10](#_6.10_Privilegios_e) podemos ver los puntos necesarios para implementar las interrupciones y excepciones según el estándar, por otra parte, en el [anexo 11](#_6.11_Entrada_Salida) podemos ver la implementación de las mismas en el simulador RARS. Y ya para finalizar podemos ver la investigación del controlador PLIC (Plataform Level Interrupt Controller) en el [anexo 12](#_6.12_PLIC).

# 5 Conclusiones de la Beca

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

Como resultados de aprendizaje se pueden destacar las ganancias intelectuales sobre un estándar ensamblador que muy probablemente sea líder en muy pocos años. Desde son sus principales instrucciones, códigos, simuladores, implementaciones. Hasta el funcionamiento de la entrada salida, los distintos niveles de privilegios, etc.

Es decir: se ha podido obtener principalmente un conocimiento mucho más profundo en los ámbitos principales del estándar RISC-V y su entorno.

## 5.2 Conclusiones Personales

El desarrollo de la beca de colaboración me ha aportado de manera personal algunos de estos puntos: conocimientos en un sector de bastante interés para una buena cantidad de empresas y compañías. Práctica adicional con la redacción de documentos y la documentación del progreso del trabajo, con esperanzas de sacarle provecho en futuros trabajos y proyectos: más inminentemente con el Trabajo de Fin de Grado (TFG). Adicionalmente me ha otorgado una experiencia laborar que puedo aprovechar para impulsar mi carrera laboral o al menos hacer por diferenciar un poco mi Currículum Vitae (CV).

# 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 [7], 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 [7] 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 [8]. 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 [7] 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 [9], 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 [10] 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 [11], 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 [11] 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 [12].

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 [13]. 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 este simulador solo tenemos que trabajar como haríamos normalmente con el editor de texto, al menos para la redacción del código. Por otro lado, cuando se depura y ejecuta el código podemos ver muy cómodamente las instrucciones que se van ejecutando, aunque la visualización de la memoria y los valores de las variables y registros son un poco incómodos de ver.

### 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 [14].

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 [15] [16], 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 [17] [18] [19], 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 [20] [21], 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 [22].

### 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.

## 6.9 Trabajo con “C” y ensamblador RISC-V con RIPES

### 6.9.1 Introducción

RIPES es un simulador que principalmente se usa para trabajar con programas escritos en ensamblador RISC-V. Sin embargo, este simulador está preparado también para ejecutar programas escritos en un lenguaje de más alto nivel como es “C”.

Para poder trabajar con este tipo de programas, en esta plataforma tendremos principalmente dos opciones: trabajar a partir del propio interfaz del simulador, siendo esta la opción más básica y cómoda, aunque también la que cuenta con menos libertades; o montar el ejecutable del programa fuera y trabajar con este en el propio simulador.

Para la realización de este documento se ha trabajado con las configuraciones de RIPES para RV32I [23], lo que también implica que el compilador que se ha usado es el especializado para este mismo caso (riscv32-unknown-elf). Nótese que de trabajar con otras configuraciones u otro compilador es posible que se necesite hacer cambios en algunas de las sentencias que se emplean en los ejemplos de este documento.

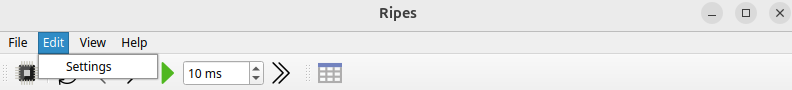
### 6.9.2 Configuración y requisitos previos

Para trabajar con programas “C” en RIPES, a diferencia de los programas ensamblador RISC-V, necesitaremos instalar a parte el compilador específico que nos permitirá crear los archivos ejecutables. Esto se puede hacer de diferentes formas: se puede hacer a través del repositorio oficial [24] donde encontramos el código fuente sobre el que crearemos todos los ejecutables, opción que no recomiendo por lo compleja y pesada que se puede hacer; se puede instalar a través de apt, siempre que contemos con esta opción; o se puede instalar una versión precompilada y añadirla a la ruta de ejecución. Esta última opción es la que se ha usado para la realización de este documento, donde los ejecutables se han sacado del repositorio no oficial [25].

Una vez instalado el compilador que vamos a usar necesitaremos configurar RIPES para que este lo use, cabe resaltar que necesitaremos para esto la ruta absoluta donde se encuentra el ejecutable.

#### 6.9.2.1 Pasos a seguir

Primero tendremos que entrar en el apartado “Settings” dentro del apartado “Edit” en la esquina superior izquierda.



Una vez abierta la pestaña tendremos que dirigirnos al apartado “Compiler”, más concretamente tendremos que ir a donde pone “Compiler path” y añadir la ruta absoluta del compilador.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Si todo se ha ido bien el apartado se iluminará de color verde indicando que ha detectado el ejecutable y está preparado para usarlo.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez hecho esto, ya lo tendremos todo listo para empezar a trabajar con programas escritos en “C”.

### 6.9.3 Trabajar con la Interfaz de RIPES

Si queremos trabajar con programas escritos en “C” usando la propia interfaz del simulador RIPES, podremos hacerlo teniendo en cuenta una serie de restricciones: los programas que hagamos tendrán que estar escritos en un único fichero; no podremos trabajar varios lenguajes al mismo tiempo, sino que todo lo que escribamos tendrá que estar escrito en “C”; si queremos tener programas más complejos que requieran de opciones adicionales al compilador, tendremos que especificar las mismas en el apartado de configuración.

Una vez dicho esto, se mostrará un ejemplo sencillo de un programa escrito en “C”, y los pasos necesarios para ejecutarlo usando este simulador.

Empezamos abriendo el apartado “Editor” donde vamos a trabajar a lo largo de todo este proceso.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Justo encima de la consola, en la pestaña izquierda, es donde tendremos el editor en el que podremos escribir el código “C”, importante seleccionar la opción “C” justo encima del editor.

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

Ahora escribimos un programa sencillo en “C”. Para este ejemplo he elegido un programa principal que suma dos números enteros y retorna el valor resultante.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Una vez hemos terminado el programa, construimos el ejecutable y lo cargamos. Esto se hace pulsando el botón con el símbolo de martillo, para simbolizar la construcción.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Si todo ha progresado de forma correcta se debería haber generado el código desensamblado a la derecha del editor, que es donde vamos a ver la ejecución de las instrucciones una a una, aunque esto no se refleje de forma directa sobre el código “C”.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Ahora podemos ir ejecutando instrucción a instrucción, añadiendo breackpoints y viendo cómo cambia el valor de los registros, o podemos ejecutar todo el código de un tirón. En este ejemplo, una vez finalizada la ejecución del programa debería aparecer por pantalla el valor de salida, que en este caso debería ser 12.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Por otra parte, si ya se cuenta con un programa escrito en “C”, este se puede cargar desde la ruta en la que se encuentre directamente al simulador. Para esto se tiene que abrir el apartado “File”, más concretamente el apartado “Load Program”.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Dentro de este apartado seleccionamos la opción “Source File” y le damos la ruta absoluta del código que queremos cargar en el simulador.

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

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

### 6.9.4 Trabajar a partir de Ejecutables

Si queremos trabajar con programas más complejos, formados de varios archivos, o incluso, si queremos mezclar varios lenguajes como puede ser ensamblador RISC-V con “C”, vamos a necesitar montar el ejecutable aparte, y cargarlo sobre RIPES para ver el comportamiento a bajo nivel del mismo. Para esto se van a mostrar a continuación algunos ejemplos con “C” y ensamblador.

Empezando por mostrar como se haría el caso anterior montando el ejecutable desde fuera, lo primero que tenemos que hacer es compilarlo. Para esto empleamos la sentencia “riscv32-unknown-elf-gcc main.c -g -c -o main.o” que nos genera el programa objeto. A partir de este podemos ver el ensamblador correspondiente a la función principal empleando la sentencia “riscv32-unknown-elf-objdump -S main.o”.

Texto

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos los objetos que necesitamos solo haría falta construir el ejecutable a partir de estos, como si se tratase de un programa “C” estándar, para este ejemplo usamos la sentencia “riscv32-unknown-elf-gcc main.o -o prog”. Si usamos la sentencia anterior a esta podemos ver también el código ensamblador, aunque a diferencia del objeto, el programa tendrá también el resto de las funciones de librería, lo que puede dificultar el encontrar las sentencias que realmente nos interesan.

Texto

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos el ejecutable ya montado solo hace falta cargarlo en RIPES. El proceso es similar al de cargar un código “C” o ensamblador a partir de un fichero ya existente. Para ello nos vamos a la interfaz “File”, “Load Program” y seleccionamos la opción “Executable (Elf) ”, le damos la ruta absoluta del programa ejecutable que queremos montar, y si todo ha ido correctamente, entonces ya podríamos simular el programa desde RIPES. Se seguirán estos mismos pasos para los siguientes ejemplos a la hora de cargar los programas en RIPES.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Pasando al siguiente nivel, ahora vamos a hacer un programa sencillo en “C” que integre también funciones escritas en ensamblador. Para esto, le damos a la función un nombre que vayamos a usar tanto en el código “C” como en el código ensamblador, y la declaramos como extern. Hay que tener en cuenta como realiza el compilador el paso de parámetros para las funciones, en el caso de RISC-V el paso de parámetros siempre que se puede, se hace mediante registros. Tomando el ejemplo anterior, si declaramos una función suma, veremos que para pasar los dos enteros se usan los registros a0 y a1, mientras que para el valor de retorno se usa el valor a1.

Texto

El contenido generado por IA puede ser incorrecto.

Por otra parte, para el código ensamblador tenemos que declarar como globales todas aquellas funciones que vamos a usar en el programa, esto se hace con la directiva del compilador “.globl”, luego solo tenemos que escribir el código asociado a la función.

Imagen de la pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos todos los códigos que necesitamos para montar el programa, tendremos que crear el programa a partir de los mismos, o hacerlo directamente. Una vez hecho solo tenemos que seguir los mismos pasos del ejemplo anterior, cargando el programa en RIPES para poder ejecutarlo.

Texto

El contenido generado por IA puede ser incorrecto.

Ya para finalizar con los ejemplos, que ocurriría si en vez de declarar una subrutina hoja en ensamblador, quisiésemos traducir una función que llama a otras funciones de “C” a ensamblador, pero solo esta función y no las que llama. Pues podemos hacer esto si dentro del código ensamblador declaramos estas funciones adicionales como externas, lo que se hace con la directiva del compilador “.extern”. Siempre, claro está, teniendo en cuenta que el paso de parámetros que usemos coincida con el que usa el compilador por defecto.

Para este ejemplo, vamos a crear un programa sencillo en “C” que realiza la suma del factorial de dos números enteros y la retorna como valor de salida. Y luego vamos a traducir la función que los suma, pero no la función que hace la operación factorial de un número entero, sino que la vamos a dejar tal cual está en el programa “C”.

Texto

El contenido generado por IA puede ser incorrecto.

Cambios para pasar la función sum\_fact a ensamblador.

Texto

El contenido generado por IA puede ser incorrecto.

Código ensamblador para la función sum\_fact.

Texto

El contenido generado por IA puede ser incorrecto.

Si ejecutamos este programa en RIPES, podemos ver que el valor de salida coincide con la operación 5! + 7!, que es la operación que hemos programado en el código anterior por lo que hemos podido comprobar que se puede mezclar código ensamblador con código “C” sin mayor complicación.

### 6.9.5 Usando fichero de enlace

Cuando trabajamos con varios ficheros de código nos puede llegar a interesar tener cierto control sobre el ejecutable resultante. Por ejemplo: en la definición y localización de las secciones, control sobre los permisos de las mismas, el valor de entrada dentro de código, la organización, etc.

Hasta este punto hemos trabajado con el fichero de enlace por defecto del compilador, sin embargo, si contamos con un fichero de enlace propio podremos especificarle al compilador que use este en vez del por defecto. El proceso de generación del ejecutable es muy similar al anterior: primero se crean los ficheros objeto compilando los ficheros de código, y una vez se tienen todos se llama al enlazador y se genera el ejecutable.

Sentencia ejemplo para generar el ejecutable (prog) a partir de los objetos (objX.o) y con el fichero de enlace definido (linker.ld):

“riscv32-unknown-elf-ld -T linker.ld -o prog obj1.o obj2.o obj2.o … “

Como se va ha mencionado anteriormente el fichero de enlace puede tener varias funciones y propósitos, es por eso que este apartado versará sobre las funciones básicas del fichero de enlace para poder empezar a trabajar, y el impacto que tienen en el impacto en el simulador, estas se pueden definir de la siguiente manera:

* Funciones para el control de la entrada
* Definición de secciones
* Regiones de memoria
* Funciones para el control de la salida

Funciones para el control de la entrada:

Hay unas cuantas funciones de control de la entrada, sin embargo, por la simplicidad de este documento solo se tratará con las tres funciones básicas: ENTRY, INCLUDE, INPUT. La funcione ENTRY define, si no se ha hecho por línea de comandos, el comienzo del código: si se cuenta con una función main se puede poner por ejemplo ENTRY(main) para que el programa empiece en el comienzo de la función main. La función INCLUDE sirve para dar rutas en las que buscar los diferentes archivos. Y ya por último, la función INPUT incluye el fichero que se defina en la función: por ejemplo se podría añadir la sentencia INPUT(“obj1.o”) que nos permite no tener que añadir el fichero obj1.o en la sentencia cuando se compila.

Declaración de secciones (SECTIONS):

Esta sección se reserva para la declaración de secciones de código, cada sección puede tener una serie de valores, como los privilegios, dirección de comienzo de la sección, y dentro de cada definición de sección de código se puede organizar el orden en el que aparece cada subsección.

Como detalle adicional, RIPES, a pesar de permitir la declaración de nuevas secciones no parece permitir el control de los privilegios, por ejemplo: el código ejecutable tendrá que estar definido dentro de la sección text, de otra forma, cuando el programa comience a ejecutar a partir de dicha sección el simulador terminara por completo la ejecución, como si de una instrucción ilegal se tratase.

Para la definición de secciones se toma el formato:

Texto

El contenido generado por IA puede ser incorrecto.

Regiones de memoria (MEMORY):

Esta es una forma sencilla de agrupar secciones, se empieza con la declaración MEMORY y se continua con el nombre, los privilegios, el origen y lo que ocupa diche región.

Para la definición se toma el siguiente formato:

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Funciones para el control de la salida:

Hay unas cuantas funciones de control de la salida, sin embargo, por la simplicidad de este documento solo se tratará con las tres funciones básicas: FORMAT, ONPUT. La funcione FORMAT define el formato que tendrá el ejecutable resultante del enlazado de los objetos, mientras que la función OUTPUT denota su nombre. Esto nos permite tener un nombre y formato por defecto en el ejecutable sin necesidad de explicitar este proceso en la línea de comandos.

Entonces, haciendo uso del ejemplo anterior, si quisiésemos cambiar y ordenar las direcciones en las que se almacena el código y los datos en memoria, podemos usar un fichero de enlace como este:

Texto

El contenido generado por IA puede ser incorrecto.

Otra opción con el uso de regiones de memoria sería:

Texto

El contenido generado por IA puede ser incorrecto.

Adicionalmente, usando el fichero de enlace el código se muestra mucho más limpio, como se puede visualizar con el objdump.

Tras estudiar las posibilidades del fichero de enlace se ha planteado la posibilidad de usar este mismo dentro del simulador RIPES, cambiando los valores de configuración del propio simulador, más específicamente cambiando la sentencia del compilador. Sin embargo, tras varias pruebas se ha demostrado la ineficiencia de este formato, a diferencia de otros entornos de simulación como “ECLIPSE”, RIPES, que no está preparado para trabajar con varios ficheros al mismo tiempo, no puede usar de manera sistemática un fichero de enlace dependiendo del momento en el que se ejecute. Se tendría que estar cambiando el fichero de enlace por cada proyecto, y a su misma vez los valores de configuración, esto sin contar que solo se podría compilar el código con el programa principal de una vez, teniendo que contar con el resto de ficheros ya compilados antes de construir el ejecutable con el botón que nos presenta el simulador, tal como se ha visto en el apartado de trabajo de “C” con RIPES desde dentro.

### 6.9.6 Resumen

En los apartados anteriores hemos visto cómo trabajar con códigos escritos en lenguaje “C”, donde hemos podido ver que hay dos formas principales de trabajar con ello.

Podemos trabajar con la interfaz que nos ofrece el simulador RIPES cargando el código "C" en su editor. Esta forma de trabajar es bastante cómoda, ideal para trabajar con programas sencillos montados en un único archivo. Sin embargo, también ofrece pocas libertades, siendo que para los programas más complejos o que se montan sobre varios ficheros no nos es suficiente.

Por otra parte, si buscamos trabajar con códigos más complejos: que mezclen ensamblador con lenguaje “C”, que estén repartidos sobre varios ficheros, etc. RIPES nos ofrece otra opción, como es crear el ejecutable por fuera y cargarlo en el simulador para que le ejecución se haga desde dentro. Esta opción puede no ser tan cómoda como la anterior, pero si buscamos salir de las limitaciones que nos impone, o buscamos una solución especializada es la única opción de la que disponemos.

Para hacer más sencillo el trabajo anterior, añado a continuación los mandatos que he empleado para generar los ejecutables en los ejemplos, con una breve descripción para cada una de estos.

Para generar el código objeto de un programa en cuestión, esté escrito en “C” o en ensamblador usamos el compilador de riscv.

“riscv32-unknown-elf-gcc cod.c -g -c -o cod.o”

“riscv32-unknown-elf-gcc cod.s -g -c -o cod.o”

La opción “-g” añade las opciones para depuración, esenciales si queremos visualizar el código “C” al que corresponden las líneas de código máquina en el visualizador (objdump) por ejemplo.

La opción “-s” nos permite generar el código ensamblador a partir de un código “C”, por ejemplo:

“riscv32-unknown-elf-gcc cod.c -s cod.s”

Donde cod.c es el código “C” y cod.s es el código ensamblador correspondiente al mismo.

Una vez tenemos todos los objetos que necesitamos para montar el programa, podemos hacerlo también usando el compilador riscv, al igual que lo haríamos con un compilador de “C” estándar como es el gcc.

“riscv32-unknown-elf-gcc main.o obj1.o obj2.o … -o prog”

También lo podemos hacer directamente sin tener que generar los objetos, al igual que lo haríamos con un compilador de “C” estándar.

“riscv32-unknown-elf-gcc main.c obj1.c obj2.c … -o prog”

Una vez hemos generado los objetos, o el programa ejecutable, como estos ya están en código máquina que no podemos entender, si queremos visualizar su contenido podemos usar la herramienta objdump con la opción -S.

“riscv32-unknown-elf-objdump -S prog”

Que nos mostrará las instrucciones máquina que conforman el programa que queremos visualizar.

Para resumir, suponiendo que tuviésemos un programa principal escrito en “C” (main.c), con dos códigos adicionales, uno en ensamblador (obj1.s) y otro escrito en “C” (obj2.s). Si quisiéramos crear el ejecutable final y visualizar su contenido, tendríamos que ejecutar las siguientes sentencias:

“riscv32-unknown-elf-gcc main.c -c -o main.o”

“riscv32-unknown-elf-gcc obj1.s -c -o obj1.o”

“riscv32-unknown-elf-gcc obj2.c -c -o obj2.o”

Para generar el ejecutable:

“riscv32-unknown-elf-gcc main.o obj1.o obj2.o -o prog”

“riscv32-unknoen-elf-ld -T linker.ld -o prog obj1.o obj2.o”

También se puede usar: “riscv32-unknown-elf-gcc main.c obj1.s obj2.c -o prog”

Para visualizar la información de los objetos y los ejecutables

“riscv32-unknown-elf-objdump -S prog”

## 6.10 Privilegios e Interrupciones en RISC-V

### 6.10.1 Aclaración Inicial

En este documento aporta una visión inicial del comportamiento del estándar frente a los niveles de privilegios sobre los que ejecuta, junto a las interrupciones y excepciones. Para esto se hace referencia en todo momento al estándar oficial (versión actual 2024) [23] [26] [1].

### 6.10.2 Introducción

Para entender el comportamiento de las interrupciones y excepciones dentro del estándar, es necesario entender primero el concepto sobre el que se fundamenta. RISC-V es un estándar que parece más orientado en un primer momento al tratamiento de las traps a distintos niveles de ejecución, que al propio tratamiento de las interrupciones que han podido generar dispositivos externos. Esto es algo curioso teniendo en cuenta que uno de los campos en los que más destaca actualmente es en el de los sistemas empotrados, donde aparentemente el control de las interrupciones externas es más prioritario que la propia gestión de privilegios.

El estándar justifica este punto de vista como un intento de dar soporte a distintos formatos de software, desde sistemas con un único nivel de ejecución, pasando por sistemas de tipo Unix, hasta sistemas de multiprocesamiento más complejos que están gestionados por distintos sistemas operativos. Está claro que para dar soporte a estos sistemas se necesita una jerarquía de privilegios, lo que nos lleva al siguiente apartado.

Diagrama, Texto

El contenido generado por IA puede ser incorrecto.La figura 1.1 muestra tres ejemplos de sistemas que están soportados por el estándar.

Figura 1.1

### 6.10.3 Niveles de Privilegios

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.El estándar tiene definidos 3 modos de ejecución, sin embargo, dependiendo del fabricante se podrían implementar otros a demanda de la necesidad. Estos niveles de privilegios son: Usuario/Aplicación, Supervisor y, Máquina; donde el modo máquina es aquel que tiene mayor nivel de privilegios, seguido por el supervisor y llegando al usuario como el que tiene la menor cantidad. Es precisamente por esto que cualquier implementación del estándar tiene que tener definido el nivel de máquina, junto a sus registros de control, siendo el resto de los modos de operación completamente opcionales.

Figura 1.2

El estándar está planteado para que cuando se trate de ejecutar una instrucción sin los permisos suficientes, se genere una excepción, entrando en una de las traps del entorno. Normalmente el código ejecuta en nivel de aplicación hasta que entra en una trap, que puede ser una llamada al sistema, o una interrupción. El tratamiento de estas traps puede ser vertical, escalado a un nivel superior de ejecución, u horizontal, manteniendo el nivel de ejecución. Esto no está definido de una manera rigurosa, sino que el propio estándar permite cierta flexibilidad en el enrutamiento de las traps.

Adicionalmente y, dependiendo del sistema que se plantee implementar, como se ha mencionado al comienzo del apartado, puede ser que se necesiten modos de ejecución adicionales. Siendo los más típicos el hypervisor para algunas implementaciones con jerarquías más complejas, y el de debug, siendo este un modo de ejecución que puede tener más privilegios que el modo máquina, donde puede reservar algunos de los registros de control e incluso partes del espacio de memoria de la máquina.

Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto.

Figura 1.3

La figura 1.3 muestra algunos ejemplos de los modos que se necesitarían implementar para algunos de los sistemas más típicos.

### 6.10.4 Registros de Control y Estado (CSR):

Cada nivel de privilegios tiene una serie de registros de control y estado con los que tiene que contar para poder trabajar de manera correcta. Donde los registros asociados a un nivel de privilegios son solo visibles por un nivel igual o superior. Estos registros de control y estado comprenden un espectro bastante amplio de funcionalidades, como puede ser: el manejo de interrupciones y excepciones, la delegación del tratamiento de las mismas, un registro de los ciclos, paginación, etc.

Para trabajar con estos registros de control y estado tenemos básicamente dos tipos de instrucciones: las que trabajan de forma atómica y nacen de la extensión “Zicsr” (las instrucciones csr), y sus derivadas. Antes de explicar el funcionamiento de las principales instrucciones del tipo “csr” para el trabajo con los registros, cabe mencionar que en este documento se adjunta un listado con los registros de control y estado contemplados por el estándar.

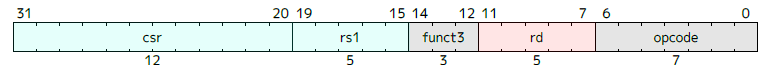
****Las instrucciones csr son instrucciones que normalmente garantizan la atomicidad de la operación, donde escriben, leen o modifican un registro de control o estado. Si se intenta acceder a un registro inexistente o del que no se tienen privilegios suficientes se levantará una excepción. Estas instrucciones pueden tomar uno de dos formatos: csrXX rd, csr, rs1, para trabajar con un registro fuente, o csrXX rd, csr, imm para trabajar con datos inmediatos.

Figura 1.4

Solo hay tres tipos de instrucciones csr: la de lectura y escritura (csrrw), la de lectura y clear (csrrc), y la de lectura y set (csrrs), tambien se cuenta con las opciones con datos inmediatos (csrrwi, csrrci, csrrsi). Para el formato de escritura se recupera sobre el registro destino (rd) el valor del registro de control (csr), y se carga sobre el mismo el valor fuente (rs1/imm). Los formatos clear y set son muy parecidos al anterior en el sentido de que la lectura se realiza de la misma manera, siendo el único cambio el no escribir sobre todo el registro de control, sino solo los bits que sea necesario: ponerlos a uno (set) o a cero (clear).

Por el código de operación que toman las instrucciones csr podríamos tener diferenciados hasta 4096 registros de control y estado posibles. Sin embargo, en el estándar hay especificados apenas unas decenas, aunque permite la definición de nuevos registros de control por parte del fabricante, siempre que no sea incompatible con la implementación del estándar ya descrita. Añadiendo a este último punto, con el tiempo han surgido algunas implementaciones no estándar para un manejo más complejo de interrupciones, que cuentan con la declaración de nuevos registros, aunque la mayor parte del control y estado se implementa mapeada a memoria, se verá en más detalles en el apartado de controladores.

Entendiendo que cada modo de ejecución tiene una serie de registros de control muy parecidos en funcionamiento al del resto de modos, solo diferenciándose en la identificación numérica y la primera letra en el nombre para identificar al modo al que pertenecen. Parece conveniente solo explicar algunos de los registros de estado y control, pero solo para el modo máquina, que son los que tienen que estar siempre implementados. Añadiendo a esto, se intentará focalizar el centro de atención en aquellos que forman parte del control de las interrupciones y excepciones.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.Empezando por el principal registro de estado y control, mstatus (machine status), este registro nos da información del sistema, donde los primeros veintidós bits tienen un propósito específico en el estándar, sin embargo, los que nos interesan son aquellos que habilitan las interrupciones y excepciones, que en este caso son los XIE donde X es el identificador del modo de ejecución.

Figura 1.5 Registro mstatus

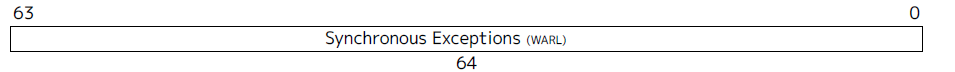
En la figura 1.5 podemos ver como este es el bit número 3, o el bit MIE (machine interrupt enable) del registro. Por otra parte, si queremos interrupciones anidadas tendremos que activar el bit MPIE (machine previous interrupt enable), donde el valor del nivel de privilegios anterior se almacena en los dos bits MPP (machine previous privilage).

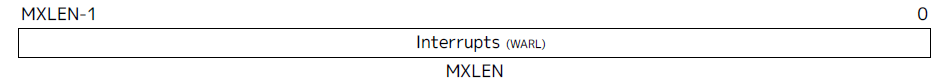
El registro mtcev ( machine trap vector interrupt) es el principal encargado de la vectorización de las interrupciones y excepciones. En este registro se guarda la dirección base sobre la que se calculará el salto cuando se produzca una interrupción o excepción. Como la dirección tiene que estar alineada a palabra ( 4 bytes ) se usan los dos bits menos significativos para indicar el modo de operación: cuando están los dos puestos a cero entonces el salto es directo y no se calcula ni diferencia nada, mientras que cuando el valor que representan es 1, al generarse una interrupción se añade sobre esta dirección el producto por cuatro del valor numérico que representa la causa de la interrupción, se verá en más detalle con el registro mcause.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto. Figura 1.6 Modos de operación del registro mtvec

Por defecto las interrupciones y excepciones se tratan en el modo máquina, sin embargo, como se mencionaba al principio del apartado, se pueden delegar a niveles inferiores. Para esto es necesario implementar dos registros de control que permiten llevar a cabo esta tarea: mideleg ( machine interruption delegataion register ) y, medeleg ( machine exception delegation register ). Nótese que para las implementaciones con modo supervisor, estos registros se tendrán que implementar de manera obligatoria.

 Figura 1.7 Registro mdeleg

**** Figura 1.8 Registro mideleg

Para el manejo de las interrupciones tenemos los registros de interrupción: mie (machine interrupt enable register ), y mip ( machine interrupt pending register ). Donde los 16 bits menos significativos de estos dos registros están asociados a valores específicos del estándar, de forma completamente simétrica. Podemos ver por ejemplo las interrupciones del timer y las interrupciones software, donde los16 bits más significativos están preparados para que el fabricante pueda particularizar la implementación a necesidad. **Interfaz de usuario gráfica, Tabla, Escala de tiempo

El contenido generado por IA puede ser incorrecto.**Figura 1.9 Imagen sacada del estándar oficial

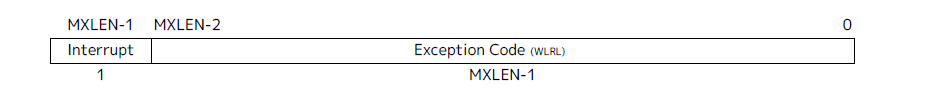
Como se había mencionado al comienzo del documento, este estándar está más pensado en el tratamiento de excepciones a distintos niveles de privilegios que en las propias interrupciones, siendo este registro el claro ejemplo. Podemos seccionar las interrupciones externas en dos, las interrupciones de timer, y el resto, como vemos en la imagen las interrupciones de timer tienen dos bits dedicados en el registro: MTIE (machine timer interrupt enable ) y, STIE ( supervisor timer interrupt enable ), siendo la misma cantidad de bits dedicados para todo el resto de interrupciones externas: MEIE (machine external interruption enable) y, SEIE (supervisor external interruption enable), lo que para muchos sistemas es claramente insuficiente. Es decir, para los periféricos externos tenemos una única línea de interrupción, donde la prioridad se trata de manera externa, por ejemplo, con un controlador.

Una vez que se interrumpe, ya sea una interrupción externa o una excepción, se almacena el contador de programa en un registro específico diseñado para esto: mepc (machine execution program counter), siendo que se usará este registro en las instrucciones de retorno de interrupción (mret, sret, uret, …), si por ejemplo, se ha generado una excepción y queremos tratarla saltando a la siguiente instrucción, entonces tendremos que sumarle cuatro al valor del registro, almacenarlo, y retornar de la interrupción.

Como se ha mencionado con el registro de vectorización, las interrupciones y excepciones tienen un valor específico que las identifica, o al menos la causa que las ha generado. Para este propósito tenemos el registro mcause (machine cause register), en el que, cuando se genera una interrupción, se almacena el valor numérico asociado a la causa de la misma, lo que se puede usar para identificar el tipo de la misma.

Las figuras 1.10 y 1.11 muestran el formato del registro y las codificaciones de las causas definidas por el estándar.

**Tabla

El contenido generado por IA puede ser incorrecto.**Figura 1.10 Estructura del registro mcause

**Tabla

El contenido generado por IA puede ser incorrecto.**

**Tabla

El contenido generado por IA puede ser incorrecto.**

Figura 1.11 Tabla con las causas definidas en el estándar.

Dentro del estándar también se ha definido la prioridad de ejecución de las excepciones síncronas, esto se puede ver en la figura 1.12 sacada directamente del estándar.**Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**Figura 1.12

Para identificar además de la causa de la interrupción, información adicional, se implementa el registro mtval (machine trap value), que está pensado como asistente para el tratamiento de las interrupciones y excepciones. Cuando se genera una interrupción o excepción este registro o bien tiene el valor 0, o bien contiene información específica de la interrupción o excepción. Por ejemplo, cuando se hace un acceso desalineado, este registro puede contener una parte de la dirección virtual que ha generado la interrupción, mientras que de otra manera solo tendríamos el punto del programa en que se generó.

Como se había mencionado antes, las interrupciones externas se desglosan en dos tipos principales, las interrupciones de timer y el resto. Entonces vemos una definición bastante completa del funcionamiento del timer en el estándar, donde también nos define dos registros de control específicos, mapeados a memoria, para el funcionamiento correcto de este dispositivo: mtime, y mtimecmp.

Los registros mtime y mtimecmp son ambos registros de 64 bits, donde el registro mtime marca el valor actual del timer, siendo un registro que por definición se tiene que ir incrementado al mismo ritmo que el del procesador. Mientras que mtimecmp contiene un valor estático que una vez alcanzado por mtime generará una interrupción de tipo timer. Estos dos registros se pueden modificar, tanto para reiniciar el timer, como para especificar tras cuanto tiempo se debe interrumpir.

### 6.10.5 Instrucciones privilegiadas adicionales

En este apartado se van a mencionar dos instrucciones que pueden ser útiles en el manejo de las interrupciones o excepciones: mret (Xret) y WFI. Una vez se he generado una excepción o interrupción para poder retornar de la rutina de tratamiento de forma sencilla, podemos ejecutar una de las instrucciones de la serie xret, donde x corresponde al modo de ejecución (mret para modo máquina), esta instrucción restaura el modo de ejecución y salta al punto en el que se encontraba cuando se generó la interrupción. Esta es básicamente la instrucción de retorno de interrupción, aunque también puede tener otros usos, como para el cambio de modo de ejecución.

La siguiente instrucción que puede ser de interés cuando se está trabajando con interrupciones es la instrucción wfi (wait for interrupt), que en este caso es directamente una instrucción que por definición se queda a la espera de una interrupción, gastando la menor cantidad de recursos posible. Para más información véase el apartado 3.3.3 del volumen II del manual de instrucciones [26].

### 6.10.6 Resumen

Para resumir el trabajo del estándar que se ha descrito en los anteriores apartados, a continuación, se describirá el proceso de un proyecto de entrada salida ejemplo. Por mantener un carácter genérico no se van a dar detalles específicos de la implementación.

Lo primero que vamos a necesitar son los modos de operación que se van a implementar, o sobre los que se van a trabajar. Por simplicidad, se va a plantea un sistema con solo dos modos de operación, el principal y obligatorio, el modo máquina (M), y un modo secundario para trabajar con menos privilegios, modo supervisor (S).

Lo siguiente será seleccionar las excepciones que se van a cubrir junto con los periféricos con los que se va a trabajar. Se tendrá que reservar un espacio de memoria para la vectorización y el tratamiento de las excepciones, para esto último, se necesitará conocer el valor de causa y calcular la vectorización sobre cada una de ellas.

Luego, suponiendo que las interrupciones se manejan en el nivel superior (M), mientras que el resto del trabajo se hace en el nivel inferior (S), y entendiendo que se inicia en el modo superior (M), se inicializarán los registros pertinentes, tanto internos del procesador como los del periférico. Dentro del esquema de los registros de control internos se tendría que seguir un orden parecido al siguiente: inicializar el registro de vectorización al valor de la dirección base + 1 (mtvec), se tendría que actualizar el registro de interrupciones para habilitar las interrupciones del periférico (mie), se actualizaría el registro que contine la dirección de salto al del inicio del programa (mepc), se inicializaría el registro de estado mstatus con el valor actualizado en los bits mpp y mpie, en este caso 1 para ambas regiones, básicamente cargar sobre este registro el valor 0x880. Y una vez inicializados los registros, y lo que se crea conveniente para el manejo del periférico, se retornará con la instrucción mret para cambiar de modo de ejecución al especificado en los registros.

En la figura 1.13 se presenta un código ejemplo que puede servir para la inicialización de los registros internos y del sistema.

Texto

El contenido generado por IA puede ser incorrecto.Figura 1.13 Código ejemplo y conceptual

A partir de este punto podríamos continuar con la ejecución del programa controlando solo aquellos registros que son internos del periférico, al menos, hasta que se genere una interrupción o excepción, donde se saltará a la dirección especificada, se cambiará automáticamente de modo de ejecución y se podrá proceder con el tratamiento de la interrupción. Normalmente, en la rutina de tratamiento tendremos que usar aquellos registros relacionados, tanto para el control de las interrupciones (mstatus, mtvec, mepc, mie, … ), como para obtener información de la misma (mcause, mtval, mip, … ).

El tratamiento tanto de interrupciones como de excepciones dependerá tanto del programador como de los propios periféricos y el uso que se les quiera dar. De hecho, este mismo ejemplo se podría extrapolar a sistemas más sencillos, con un único nivel de privilegios (M), o a sistemas más complejos, con varios niveles de privilegios (M, S, U), donde se tendrían que manejar los registros pertinentes al modo de operación que se vaya a encargar de las interrupciones.

### 6.10.7 Información Adicional (Controladores)

Para una mayor compatibilidad con las interrupciones externas, se han desarrollado una serie de controladores completamente compatibles con el estándar. Aunque, hay que tener en cuenta que no son completamente oficiales, a pesar de estar comprendidos en el repositorio oficial. Estos controladores son: PLIC (Plataform-Level Interrupt Controller) [27] y CLIC, el actualmente ACLIC (Advanced Core Local Interrupt Controller) [28].

Donde están planteados para una gestión más eficaz de las interrupciones, como añadir una jerarquía de prioridad, un mejor control de algunas de las interrupciones interprocesador, como cuando se tienen que manejar varios dispositivos tipo timer al mismo tiempo, y controlar el flujo de las interrupciones decidiendo en que contexto se va a interrumpir, por ejemplo, para multiprocesadores.

### 6.10.8 Listado de Algunos de los Registros de Control y Estado

Las siguientes imágenes hacen referencia al apartado 2.2 dentro del estándar, manual de instrucciones segundo volumen, donde se listan los registros de control descritos en el mismo.

Tabla

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Imagen de la pantalla de un celular de un mensaje en letras blancas

El contenido generado por IA puede ser incorrecto.

Imagen de la pantalla de un celular de un mensaje en letras blancas

El contenido generado por IA puede ser incorrecto.

Un periódico con texto

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

### 6.10.9 Terminología Usada

Hart Hardware threads

OS Operating System

ABI Aplication Binary Interface

AEE Aplication Execution Environment

SBI Supervisor Binary Interface

SEE Supervisor Execution Environment

HBI Hypervisor Binary Interface

HEE Hypervisor Execution Environment

WARL Write Any / Read Legal

PMA Physical Memory Attributes

## 6.11 Entrada Salida mediante interrupciones con RARS

### 6.11.1 Introducción

RARS es un simulador que principalmente se usa para trabajar con programas simples escritos en lenguaje ensamblador del estándar RISC-V [1]. Sin embargo, este programa está adaptado para poder trabajar con excepciones y entrada salida por interrupciones, donde tendremos una serie de herramientas y registros de control para llevar a cabo esta tarea.

En este documento se van a detallar los distintos pasos necesarios para poder trabajar con excepciones e interrupciones en RARS. Para esto, se ha trabajado principalmente con las características RV32IN [23] [29], donde se entiende que el propio simulador da soporte a otras extensiones, teniendo listadas las instrucciones en la propia documentación interna del simulador.

### 6.11.2 Registros de Control e Instrucciones útiles

Para trabajar con excepciones e interrupciones hay que conocer los distintos registros de control del simulador y, como se puede trabajar con ellos. Es por esto, que este apartado se va a dedicar a algunos de los registros de control más importantes y sus funciones, junto a las instrucciones dedicadas para trabajar con los mismos.

Empezando por los registros de control, en RARS tenemos un listado bastante completo. Como se ve en la imagen a continuación cada uno de ellos, a parte del nombre, tiene asignado un identificador numérico que también podremos usar para referenciarlos.

**Tabla

El contenido generado por IA puede ser incorrecto.** Tabla

El contenido generado por IA puede ser incorrecto.

#### 6.11.2.1 Registros

* ustatus: Principal registro de control de interrupciones y excepciones. Si se quieren habilitar las interrupciones o las excepciones se tiene que cambiar este registro, también, cuando se genera una interrupción este cambia automáticamente para inhibir las interrupciones y excepciones.
* uie: Registro secundario para manejo de interrupciones, a través de este registro puedes inhibir o permitir distintos tipos de interrupciones, es decir, puedes especificar qué tipo de interrupción se puede generar.
* utvec: Registro en el que se almacena la dirección de comienzo del handler, que en este caso es único tanto para interrupciones como excepciones, dirección a la que se saltará automáticamente al generarse una excepción o interrupción.
* uepc: Cuando se genera una excepción o interrupción se almacena en este registro el valor de retorno, es decir, la dirección de la instrucción que se estaba ejecutando cuando se generó la interrupción o excepción.
* ucause: En este registro se almacena el tipo de interrupción o excepción que se ha generado, donde cada una tiene un código específico que las identifica.
* utval: En este registro se almacena un valor adicional a la interrupción que se ha generado, y no necesariamente condicionado con el tipo de interrupción.
* uip: Muestra si hay alguna interrupción pendiente y de que tipo es esta, usando el identificador de la misma.

Una vez conocidos los registros de control, necesitaremos un medio por el cual manipular sus valores asociados. Aquí es donde entran en juego las instrucciones específicas del estándar. Aun cuando ya están ya están descritas en el manual oficial [23], se dará a continuación un breve resumen de funcionalidad.

#### 6.11.2.2 Instrucciones

Las instrucciones csr son instrucciones especializadas para la manipulación de los registros de control, puesto que garantizan la atomicidad en su ejecución. Estas instrucciones siguen el formato: csr… rd, fcsr, rs1 o para las variantes con datos inmediatos, csr… rd, fcsr, imm. Véase más información en el manual de instrucciones del estándar [23].

* csrrc (Read/Clear): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 0 todo bit que se encuentre activo en el registro rs1.
* csrrci (Read/Clear Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 0 los bits indicados por el dato inmediato (imm).
* csrrs (Read/Set): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 1 todo bit que se encuentre activo en el registro rs1.
* csrrci (Read/Clear Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control pone a 1 los bits indicados por el dato inmediato (imm).
* csrrw (Read/Write): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control escribe el valor actual del registro rs1.
* csrrwi (Read/Write Immediate): Carga el valor del registro de control (fcsr), al registro destino (rd). Y, en el registro de control escribe el valor definido por el dato inmediato (imm).
* uret: Instrucción que nos permite salir de manera automática de la rutina de tratamiento de interrupciones, es decir, reactiva las interrupciones y excepciones y retorna a la instrucción que se estaba ejecutando cuando se produce la interrupción o excepción.

Como ejemplo sencillo, si queremos poner el valor 1 en el registro de control uie, podemos hacerlo con la instrucción csrrwi x0, uie, 1. De todas formas, en los siguientes apartados se mostrarán ejemplos que muestran el uso práctico de estos registros.

### 6.11.3 Excepciones

Las excepciones son fundamentales cuando estamos tratando con la detección de errores, y aun cuando muchos simuladores no las implementan por simplicidad, RARS implementa 7 tipos distintos de excepciones: instruction address misaligned (0), instruction access fault (1), illegal instruction (2), load address misaligned (4), load access fault (5), store address misaligned (6), store access fault (7), and environment call (8)

Adicionalmente, trabajar con excepciones en RARS es relativamente sencillo, solo hay que configurar el valor de los registros de control utvec, para la dirección del handler, y ustatus para que se puedan activar las excepciones. La dificultad radica en la implementación del handler, lo que se tiene que hacer completamente por software.

Para poner un ejemplo sencillo, si queremos un programa cuya manera de responder ante las excepciones sea saltando a la siguiente línea lo podemos hacer de la siguiente forma:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Este programa comienza inicializando los registros de control ustatus y utvec, donde el bit 1 del registro ustatus es el que habilita las excepciones e interrupciones, y el handler es donde se encuentra la rutina de tratamiento. Esta primera secuencia será común para la mayoría de los programas que trabajen con excepciones o interrupciones.

****

Por otro lado, como se ha decidido implementar un programa que continúe ejecutando en la siguiente instrucción, para la implementación del handler se recoge el valor del registro uepc, que, en este caso, contiene la dirección de la instrucción que generó la excepción, le suma 4 (tamaño de palabra) y retorna con la instrucción uret. Al valor del registro uepc se le suma 4 puesto que si se retornara tal cual del handler nos encontraríamos en un bucle infinito donde la instrucción que generó la excepción volvería a hacerlo. Como veremos en el siguiente apartado, para las interrupciones no se pasa a la siguiente instrucción, sino que se retorna a la que se ha visto interrumpida.  
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Este ha sido un ejemplo sencillo, sin embargo, se puede hacer un handler tan complejo como se necesite para cada caso, inclusive se puede diferenciar entre los distintos tipos de excepciones a través del su código. Para este caso, cuando se salta al handler, el registro ucause toma el valor 5 puesto que se hace un acceso ilegal en la línea 7, en caso de haberse generado otro tipo de excepción el código hubiese sido distinto.

### 6.11.4 Entrada/Salida por Interrupciones

Siguiendo con el apartado anterior, está claro que RARS tiene implementado un sistema de interrupciones, entonces, si lo que queremos es trabajar con las interrupciones externas del simulador, es decir, la entrada salida mediante interrupciones; RARS nos ofrece una serie de herramientas ya configuradas para esta tarea.

Dentro del simulador tenemos un total de 4 herramientas ya implementadas, de entre las cuales para entrada salida mediante interrupciones podemos usar solo 3, como son: un teclado y un display, un timer, y un teclado hexadecimal. En este apartado nos vamos a centrar solo en el timer y en el teclado y display, que ya son suficientes para una visión bastante completa del entorno. Nótese que los ejemplos también están simplificados suponiendo siempre o la mejor de las situaciones o la segunda mejor.

El tratamiento de las interrupciones es muy parecido a como se hacía el de las excepciones, la única diferencia es que ahora, también se tienen que inicializar otros registros de control, como es el uie, o los propios registros de control de las herramientas.

Comenzando con el timer: este dispositivo es muy sencillo en el sentido de que solo tiene una tarea, que es interrumpir una vez pasado el umbral de tiempo establecido.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.Las interacciones entre los programas y los dispositivos se hacen mayormente mediante MMIO, con una serie de direcciones reservadas para los registros de control internos del dispositivo u otros. Para el caso del timer, hay dos palabras reservadas donde se almacena el valor en milisegundos del umbral que hay que superar para que se active la interrupción externa. Es por esto que cuando inicie el programa una de las tareas que tendremos es la de almacenar en estas direcciones el valor que busquemos. Nótese que no hay manera de reiniciar el proceso del timer de forma automática, una vez que ha interrumpido, para volver a interrumpir se necesitará reiniciar de forma manual la herramienta.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En este ejemplo, se busca que el programa se mantenga en un bucle infinito y cuando el timer interrumpa el handler salte a otro bucle infinito distinto al primero. En este caso se ha entendido que el timer solo interrumpe una vez, es por esto que no se reestablecen los registros de control. Para la configuración inicial de este dispositivo, tenemos que activar el bit 4 del registro de control uie y adicionalmente, tendremos que almacenar en las dos palabras reservadas el valor del umbral en milisegundos en el que esperamos que se genere la interrupción: 32 bits menos significativos dirección 0xFFFF0020, 32 bits más significativos dirección 0xFFFF0024.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Aumentando un poco el nivel de dificultad con respecto al ejemplo anterior, tenemos la herramienta del teclado y display, que nos ofrece un apartado de recepción y otro de transmisión.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En un principio, la manera de trabajar con esta herramienta es muy parecida a como se hace con el timer: se tiene tanto para el teclado como para el display una dirección reservada que actúa como registro de control (0xFFFF0000 para recepción y 0xFFFF0008 para transmisión) donde los dos únicos bits que nos interesan son los menos significativos. El bit 0 de este registro de control es conocido como ready bit, y es el que nos indica si la recepción o transmisión están disponibles; por otra parte, el bit 1 es el que permite el trabajo mediante interrupciones, si lo activamos se podrán generar interrupciones cuando el ready bit se active. Y al igual que con los registros de control, tendremos otras dos direcciones reservadas para el envío y recibo de los caracteres (0xFFFF0004 para recepción y 0xFFFF000C para transmisión), si por ejemplo queremos que aparezca la “a” en el display tendremos que almacenar en la dirección 0xFFFF000C el 61 decimal (“a” en ascii), y esperar a que termine la operación.

Como primer ejemplo, hemos creado un programa que solo solicita interrupciones a través del teclado. Texto

El contenido generado por IA puede ser incorrecto.

Primero inicializamos los registros de control necesarios para trabajar con interrupciones, aunque sea solo para la recepción: activamos el bit 8 del registro de control uie, que es el que permite las interrupciones de esta herramienta, y luego activamos también el bit de interrupciones del registro de control de la herramienta para las recepciones (bit 1 dirección 0xFFFF0000).

Texto

El contenido generado por IA puede ser incorrecto.

Una vez que hecho esto entramos en un bucle infinito hasta que se active una interrupción, para ese momento no es necesario comprobar el ready bit del registro de control de recepción puesto que sabremos que si se ha interrumpido es porque está activo. Cuando leemos el carácter del registro de datos de recepción (dirección 0xFFFF0004) automáticamente se actualizará el registro de control del mismo, dándose la misma situación para la transmisión.

Una vez obtenido el carácter lo enviamos para la transmisión, en este ejemplo por simplicidad no se comprueba si la transmisión está disponible o no, sino que simplemente se asume, después de esto se retorna de la rutina de tratamiento.

Texto

El contenido generado por IA puede ser incorrecto.

Como problemáticas o inconvenientes a destacar, tenemos: ni la recepción ni la transmisión pueden almacenar más de un carácter al mismo tiempo, por ejemplo, si se pulsa otra tecla del teclado antes de haber leído la anterior esta se pierde; las interrupciones solo se generan cuando cambia el estado del dispositivo, es decir cuando cambia el bit de ready, para las recepciones no es algo tan problemático, sin embargo, para la transmisión aún cuando se cambia el registro de control no se interrumpe, puesto que aunque esté disponible para interrumpir, como no cambia el estado no comprueba si puede interrumpir.

Para el último ejemplo se van a jugar con la recepción y transmisión por interrupciones, teniendo en cuenta la limitación anterior, en este caso se ha diseñado el programa para que compruebe si el bit de ready está activo en la transmisión, en caso de estarlo almacena el carácter directamente, y solo si no está disponible activa las interrupciones. También, cuando se recibe una interrupción de transmisión se desactivan las interrupciones para evitar que esté constantemente interrumpiendo. En un programa más completo se usaría un almacenamiento de buffers para evitar este problema, sin embargo, en este caso por simplicidad no se ha implementado.

**Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**

Como se puede ver en el código, en este caso la complejidad del programa radica en la rutina de tratamiento de interrupción, que tiene que diferenciar entre los distintos casos posibles, lo que es gracias al registro de control utval, identificador 67, que dependiendo de si se ha generado una interrupción de recepción o de transmisión toma un valor distinto (0x40 para la transmisión y 0x80 para la recepción ). En este código para las direcciones de los registros de control de los dispositivos, se han descrito a partir de la dirección 0xFFFF0000, calculando su desplazamiento, añadiendo esta dirección en el registro s0 por simplicidad.

### 6.11.5 Resumen

RARS es un simulador que nos permite trabajar con excepciones junto a entrada salida por interrupciones. Un recurso del que simuladores como RIPES carecen. Para trabajar con las interrupciones en RARS hay que conocer y manejar los registros de control de los que dispone el simulador. Aunque es verdad que hay partes que son comunes independientemente del tipo de interrupción o de excepción.

Para activar de manera general las interrupciones y excepciones, tendremos que activar el bit 0 del registro ustatus, esto, una vez que hayamos cargado la dirección de comienzo de la rutina de tratamiento en el registro de vectorización utvec. Lo que se puede hacer con las siguientes líneas de código:

****En este caso suponemos “handler” como la etiqueta asociada al comienzo de la rutina de tratamiento de interrupciones.

Por otra parte, cuando se genera una interrupción, para identificar la misma podremos usar los registros de control utval y ucause, donde ucause nos da el código asociado al tipo de interrupción que se ha generado, y utval contendrá el valor asociado a la interrupción misma.

Para volver de la rutina de tratamiento de interrupciones podremos usar la instrucción “uret”, aunque siempre entendiendo que se vuelve a la instrucción que ha sido interrumpida o a aquella que ha generado la excepción, y no a la siguiente.

Y ya si se quiere trabajar con entrada salida por interrupciones, habrá que tener en cuenta cuales son las características del dispositivo o herramienta que vamos a usar: si tiene registros de control internos y como se configuran, cual es el tipo de interrupción que genera y que bit del registro de control uie se tiene que activar para que se puedan generar interrupciones, etc. En el ejemplo del teclado y display, que se ha visto en el apartado de interrupciones, veíamos como era necesario activar el bit 8 del registro de control uie, y mirar las direcciones asociadas a los registros de control internos del dispositivo, sin embargo, para el timer, se tenía que activar el bit 4 del registro de control uie y especificar el umbral sobre el que queríamos que la herramienta interrumpiese.

## 6.12 PLIC

### 6.12.1 Introducción

Platform-Level Interrupt Controller o PLIC [30], es la definición de un controlador de interrupciones descrito para el estándar RISC-V [1], donde habilita un control más definido de las interrupciones externas y de los periféricos, o fuentes de interrupción. Sin embargo ,incluso cuando este controlador añade algo de complejidad a la implementación general, todavía es una implementación sencilla en comparación a otras del entorno como CLIC o APLIC.

Este controlador consta de dos partes principales: un Interruption Gateway, y el CORE. Dentro del Interruption Gateway se controla el flujo de interrupción de los periféricos. Mientras que en el Core se maneja el proceso de interrupción y e destino de la propia interrupción en sí misma, dentro del estándar entendemos como destino u objetivo al contexto de ejecución dentro de un mismo HART (Hardware Thread), que a su vez se puede definir como un nivel de privilegios dentro de ese HART: por ejemplo, el modo máquina y modo supervisor.

La imagen 1.1, sacada de la propia definición oficial, muestra una representación del direccionamiento de las interrupciones a los diferentes contextos con el controlador.

Diagrama

El contenido generado por IA puede ser incorrecto.Imagen 1.1 (Figura 1 dentro de la definición oficial [27])

### 6.12.2 Flujo de las Interrupciones

El proceso de trabajo de este controlador se puede seguir de la siguiente manera: Primero el Gateway recibe una interrupción global de un periférico y la procesa, transformándola en una única notificación de interrupción que pasa al CORE. Una vez ha llegado al CORE este la procesa, y manda una notificación de interrupción a todos los objetivos (contextos) cuyo umbral de interrupción quede por debajo del valor de prioridad del periférico que ha interrumpido, entonces, uno y solo uno de los contextos reclamará la interrupción (el primero), siguiendo siempre una jerarquía de prioridades ente distintos periféricos, y cuando haya terminado de procesarla mandará un mensaje al de finalización al Gateway que podrá, en caso de ser necesario, procesar otra señal de interrupción de ese mismo periférico.

En la imagen 2.1, sacada del estándar oficial, podemos ver un pequeño diagrama que el flujo del proceso de una interrupción.

Diagrama

El contenido generado por IA puede ser incorrecto. Imagen 2.1 (Figura 2 dentro de la definición oficial [27])

### 6.12.3 Componentes y Adicionales

Para todo el proceso de interrupción y de gestión de los periféricos necesitaremos una serie de registros o almacenes donde guardar la información de contexto, y para controlar el flujo y señalizar mensajes, por ejemplo: cuando un contexto quiere reclamar una interrupción o señalizar la terminación de la misma.

#### 6.12.3.1 Interrupt Priority regs

Es necesario uno por cada periférico, en estos se guarda el nivel de prioridad de cada uno de estos periféricos.

#### 6.12.3.2 Interrupt Pending bits

En estos registros se reserva un bit por cada periférico, que se activa en caso de que el periférico en cuestión esté esperando por tener su petición de interrupción atendida.

#### 6.12.3.4 Interrupt Enable regs

Estos son registros específicos de cada contexto que marcan la habilitación de las interrupciones de todos los periféricos.

#### 6.12.3.5 Priority Thresholds

Este es un registro que se establece por cada uno de los contextos del procesador, siendo aquel que marca el umbral de interrupción de dicho contexto.

#### 6.12.3.6 Interrupt Claim and Completion register

Este es un registro específico por cada contexto del procesador, donde su uso es para especificarle al controlador el reclamo de las peticiones de interrupción y a su misma vez la finalización del tratamiento de las mismas.

### 6.12.4 Conclusión

Como conclusión de este apartado podemos entender que de una forma sencilla con este controlador se puede suplir una de las principales desventajas del trabajo con la entrada salida en el estándar RISC-V: el bajo control de las interrupciones externas. Ahora con esto se puede manejar un sistema de prioridades de interrupción junto a un mejor direccionamiento y control de las mismas.

# 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] | QEMU, «QEMU System Emulator Targets / RISC-V System emulator,» [En línea]. Available: https://www.qemu.org/docs/master/system/target-riscv.html. |
| [7] | mortbopet, «Repositorio Oficial de RIPES,» [En línea]. Available: https://github.com/mortbopet/Ripes. |
| [8] | Fuse, «FUSE,» [En línea]. Available: https://github.com/AppImage/AppImageKit/wiki/FUSE. |
| [9] | TheThirdOne, «Rars Repository,» [En línea]. Available: https://github.com/TheThirdOne/rars. |
| [10] | TheThirdOne, «RARS Releases,» [En línea]. Available: https://github.com/TheThirdOne/rars/releases/tag/v1.6. |
| [11] | andrescv, «/andrescv/jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter. |
| [12] | «Jupiter,» [En línea]. Available: https://jupitersim.gitbook.io/jupiter/es. |
| [13] | hm-riscv, «/hm-riscv/vscode-riscv-venus,» [En línea]. Available: https://github.com/hm-riscv/vscode-riscv-venus. |
| [14] | «Visual Studio Marketplace RISC-V Venus Simulator,» [En línea]. Available: https://marketplace.visualstudio.com/items?itemName=hm.riscv-venus. |
| [15] | ESEO-Tech, «/ESEO-Tech/emulsiV,» [En línea]. Available: https://github.com/ESEO-Tech/emulsiV. |
| [16] | ESEO-Tech, «/ESEO-Tech/emulsiV/doc,» [En línea]. Available: https://eseo-tech.github.io/emulsiV/doc/. |
| [17] | [En línea]. Available: https://creatorsim.github.io/creator/. |
| [18] | [En línea]. Available: https://github.com/creatorsim/creator. |
| [19] | [En línea]. Available: https://creatorsim.github.io/creator/. |
| [20] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V. |
| [21] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki. |
| [22] | [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation. |
| [23] | R.-V. International, «riscv-unprivileged.pdf,» [En línea]. Available: https://drive.google.com/file/d/1uviu1nH-tScFfgrovvFCrj7Omv8tFtkp/view. |
| [24] | riscv-collab, « riscv-collab/riscv-collab,» [En línea]. Available: https://github.com/riscv-collab/riscv-gnu-toolchain. |
| [25] | stnolting, «riscv-gcc-prebuilt,» [En línea]. Available: https://github.com/stnolting/riscv-gcc-prebuilt. |
| [26] | RISC-V International, «riscv-privileged.pdf,» [En línea]. Available: https://drive.google.com/file/d/17GeetSnT5wW3xNuAHI95-SI1gPGd5sJ\_/view?usp=drive\_link. |
| [27] | P. S. H. Committee, «RISC-V Platform-Level Interrupt Controller Specification,» [En línea]. Available: https://drive.google.com/file/d/1at94PNJl4v2eAsKIwKOsZWBxsVcY2U2F/view. |
| [28] | riscv-aclint, «RISC-V Advanced Core Local Interruptor Specification,» [En línea]. Available: https://github.com/riscvarchive/riscv-aclint/blob/main/riscv-aclint.adoc. |
| [29] | RISC-V International, «riscv-spec-2019,» [En línea]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf. |
| [30] | A. Chang, «riscv/riscv-plic-spec,» [En línea]. Available: https://github.com/riscv/riscv-plic-spec. |
| [31] | mortbopet, «/Ripes/Docs,» [En línea]. Available: https://github.com/mortbopet/Ripes/tree/master/docs. |
| [32] | andrescv, «Repositorio Oficial Jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter. |
| [33] | «Guía Instalación WebRISC-V,» [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation. |