Informe Final: Beca de Colaboración

AUTOR: BRUNO BURGOS KOSMALSKI

Índice

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.

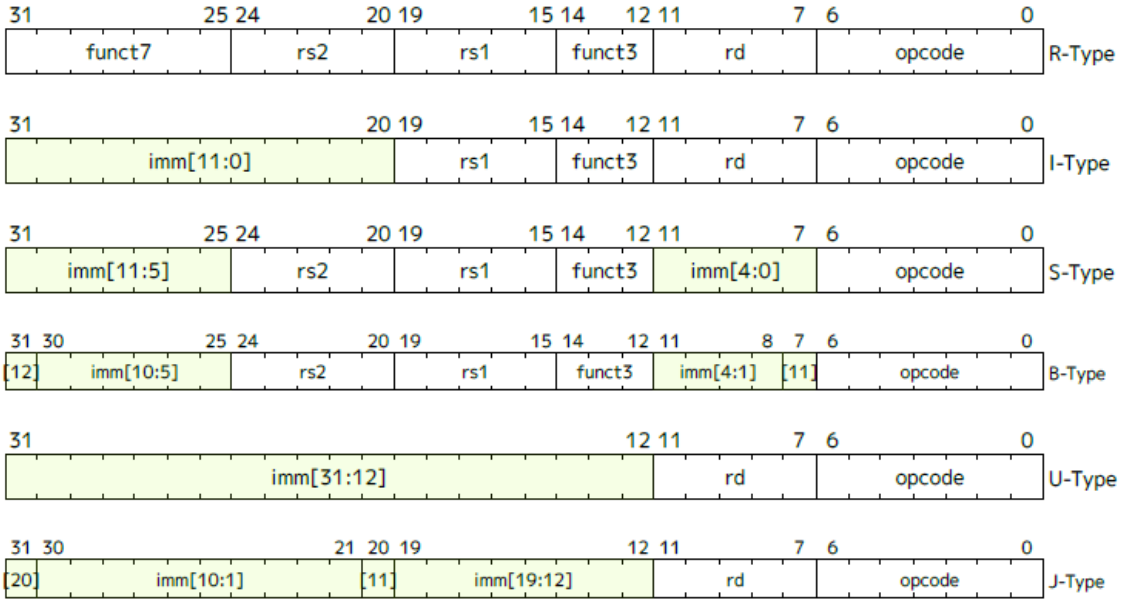

Figura 1.1

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.

2.3 Expectativas Iniciales

Expectativas iniciales depositadas sobre el desarrollo de la beca junto a los resultados de la investigación. Para dar un contraste con la conclusión de la beca. No necesariamente un punto demasiado largo.

Inicialmente las intenciones y expectativas que había sobre la beca han cumplido.

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.

Des 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, si que se descalificó aquellos cuyo proceso de instalación se consideró demasiado complejos, casos como *"SPIKE"* (añadir referencia); aquel hardware propietario y aquellos que tratan de imitar un entorno hardware, casi como máquinas virtuales, diferenciándose de los simuladores. En este último caso podemos poner como ejemplo al entorno *"QEMU"*.

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

1. RIPES
2. RARS
3. Jupiter
4. RISC-V Venus Simulator
5. EmulsiV
6. Creator
7. WebRISC-V
8. Eclipse RISC-V

Para más información sobre cada uno de estos simuladores véase el apartado anexos de este documento.

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.

A continuación se añade un resumen con las ventajas e inconvenientes de cada uno de estos 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

Tabla con las ventajas desventajas:

- Ripes: ...
- RARS:...
- Jupiter:...
- RISC-V Venus Simulator: ...
- EmulsiV: ...
- Creator: ...
- WebRISC-V: ...
- Eclipse RISCV: ...

asda

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, y en la mayoría de los casos por la simplicidad de la implementación. Por otra parte otros simuladores como Venus y Eclipse RISC-V, incluso cuando tenían la baza de que los estudiantes estuviesen más acostumbrados a estos entornos, no ha sido suficiente para contrarrestar las limitaciones de estos sistemas.

Mientras que RIPES tiene una interfaz más completa y se puede trabajar con el con un poquito más de libertad. Algunos puntos, especialmente el de la entrada salida dejan mucho que desear en comparación a RARS. Que tiene implementados registros de control junto a interrupciones y excepciones. Aunque está un poco desfasado.

4 Investigación Adicional

Este apartado tratará algunos puntos adicionales sobre los que se puso el foco en la investigación, en algunos casos a partir de las propias implementaciones dentro de los simuladores y su conformidad con respecto al estándar actual.

4.1 Compiladores y Trabajo con varios ficheros de código

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

4.2 Entrada Salida

Explicación de la entrada salida brevemente y su definición dentro del estándar: registros de control, instrucciones especializadas (Zcsr), controladores de

interrupciones para un mejor control de prioridad. Mención especial al simulador RARS y como implementa este las interrupciones y excepciones, también al intento de implementarlo sobre el simulador que se ha escogido junto a las limitaciones de RARS para defender esta decisión.

5 Conclusiones de la Beca

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

5.1 Resultados de Aprendizaje

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

5.2 Conclusiones Personales

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

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 [5], 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.

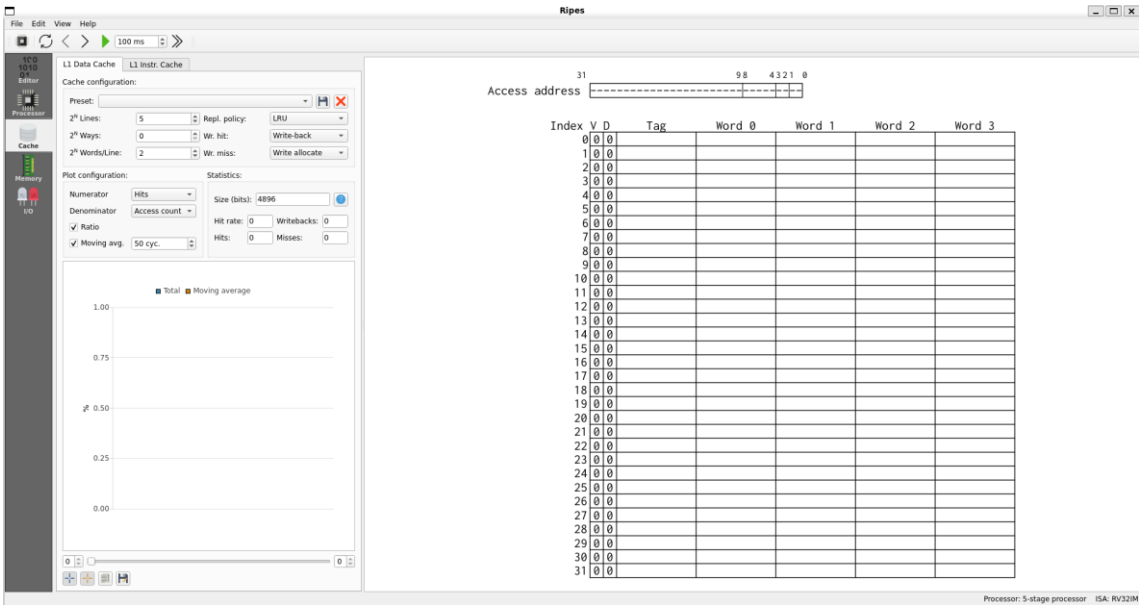

Figura 1.1

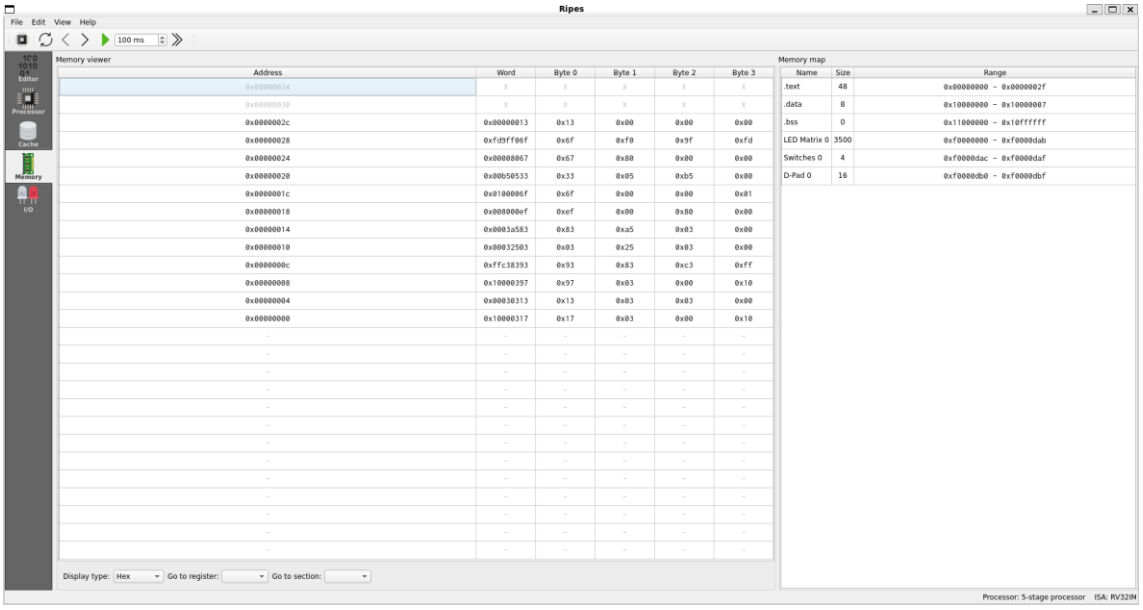

Figura 1.2

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.

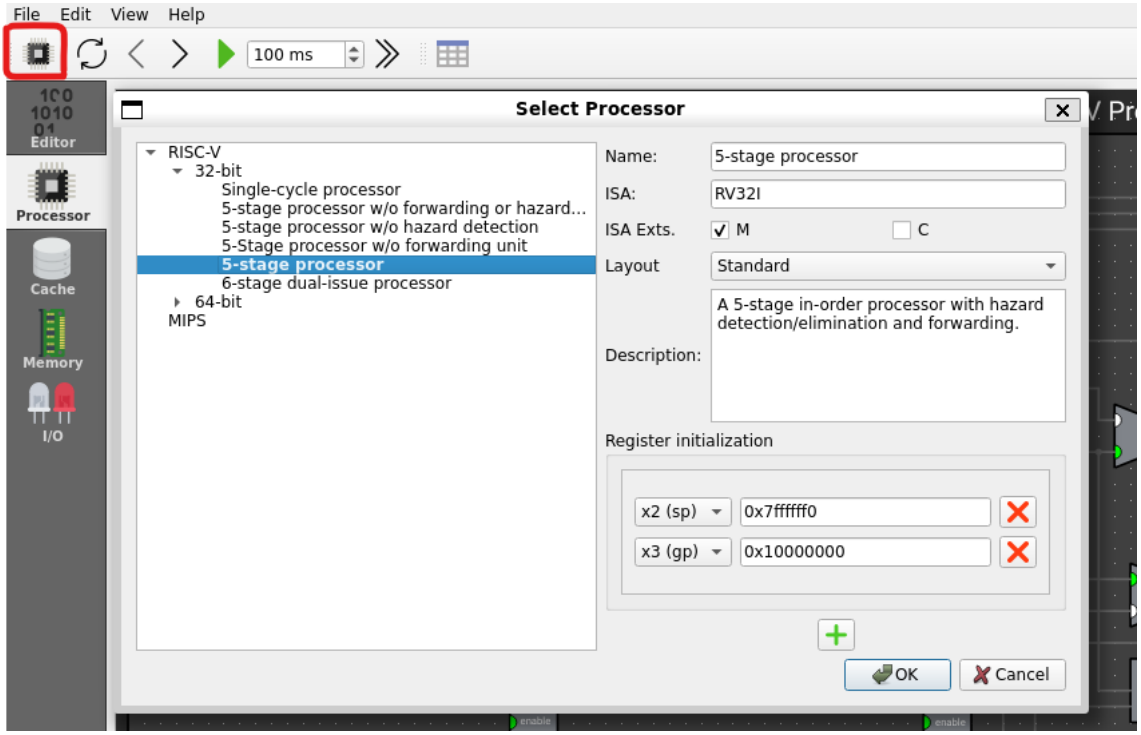

Figura 1.3

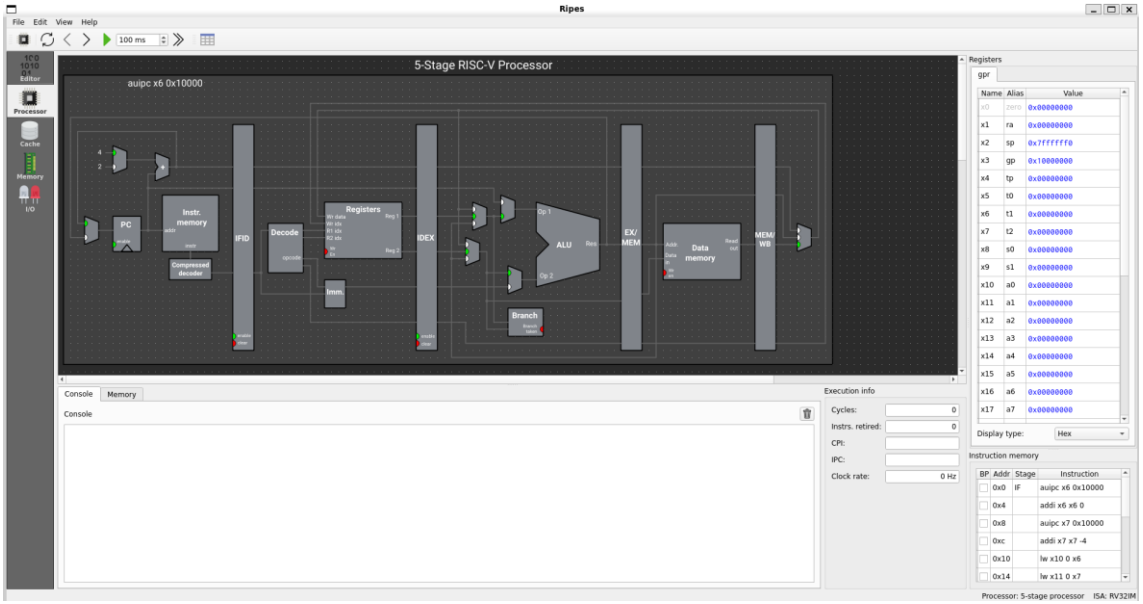

Figura 1.4

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.

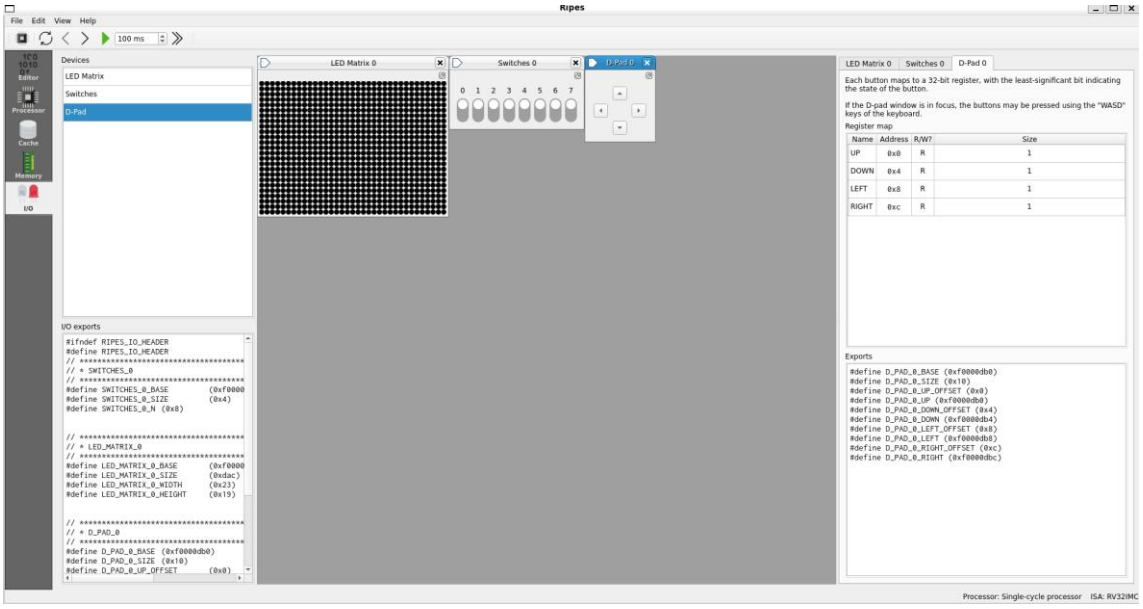

Figura 1.5

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.

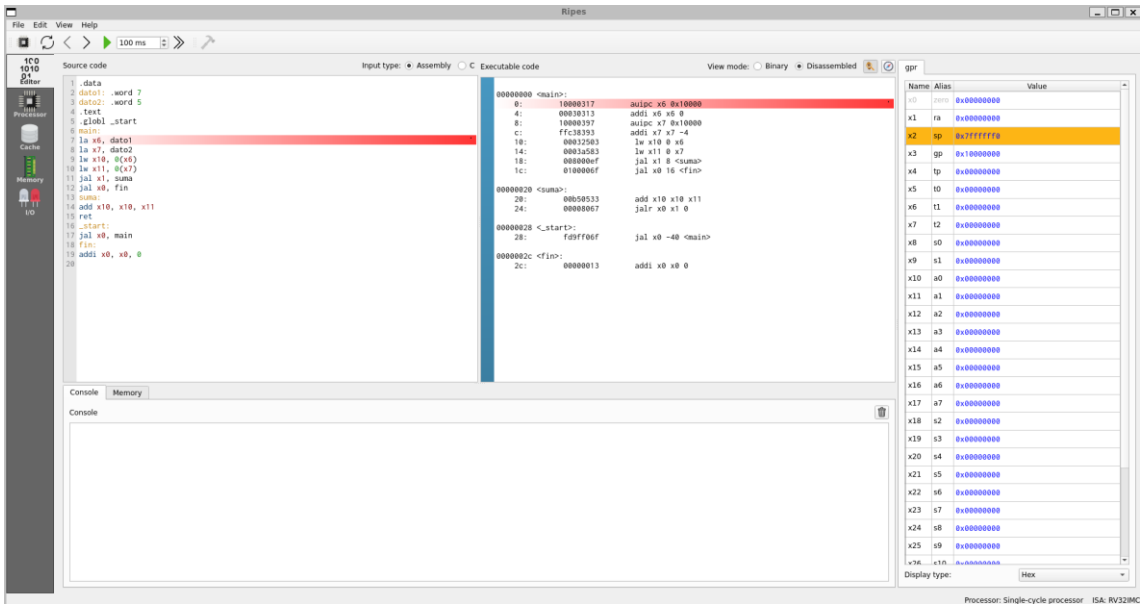

Figura 1.6

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 [5] 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.

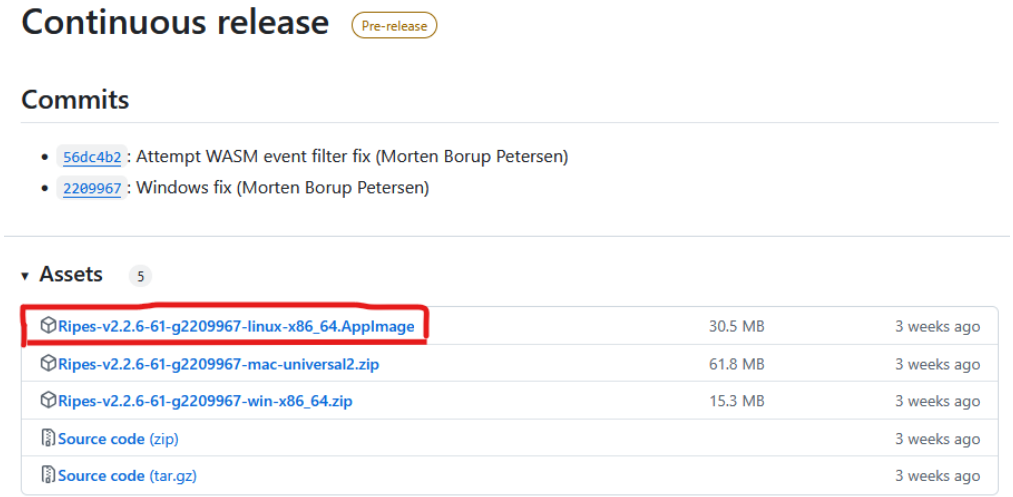

Figura 2.1

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.

```
bruno@ABURKOS:~/riscv/ripes$ ls
Ripes-v2.2.6-61-g2209967-linux-x86_64.AppImage
bruno@ABURKOS:~/riscv/ripes$ ./Ripes-v2.2.6-61-g2209967-linux-x86_64.AppImage
dlopen(): error loading libfuse.so.2

AppImages require FUSE to run.
You might still be able to extract the contents of this AppImage
if you run it with the --appimage-extract option.
See https://github.com/AppImage/AppImageKit/wiki/FUSE
for more information
```

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 [6]. 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 [5] 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.

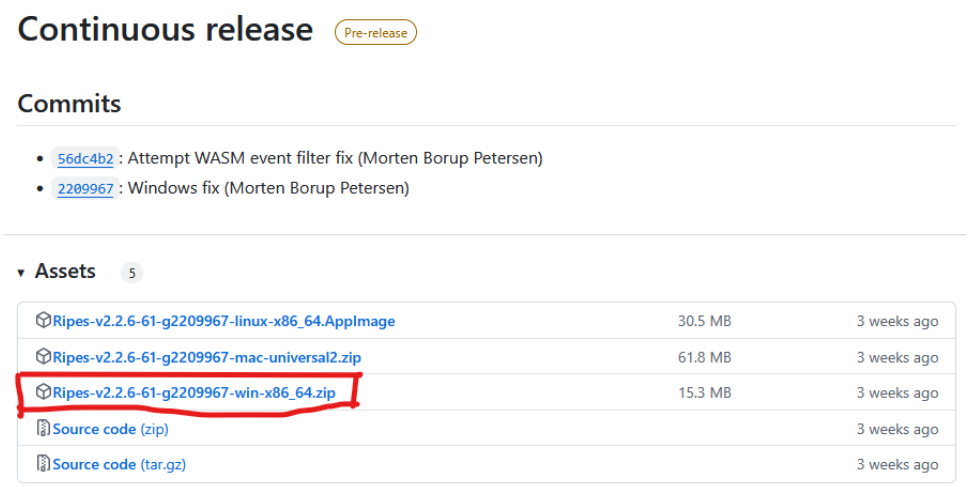

Figura 2.3

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”.

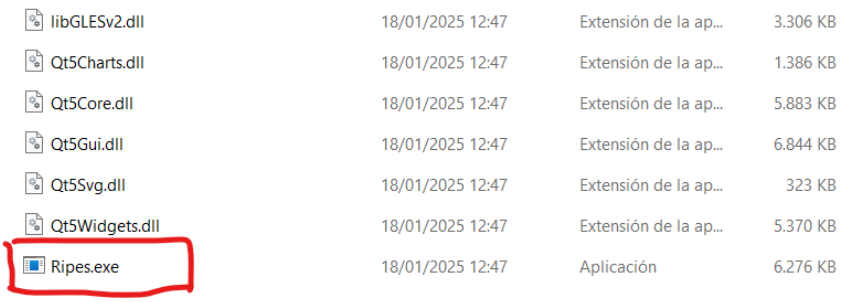

Figura 2.4

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 [7], 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.

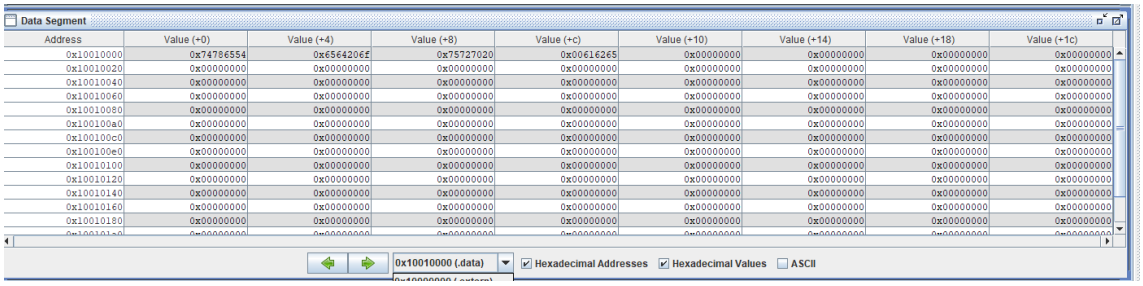

Figura 1.1

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.

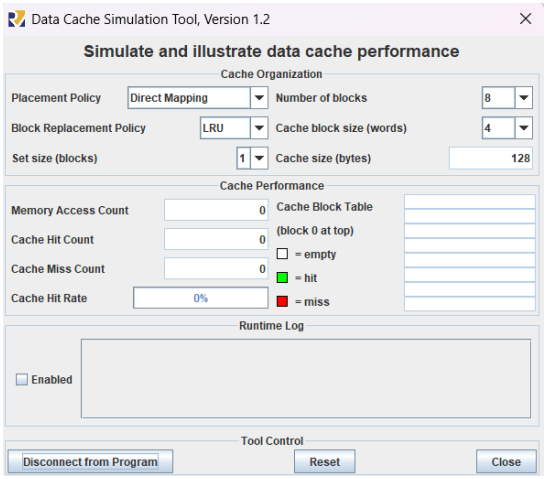

Figura 1.2

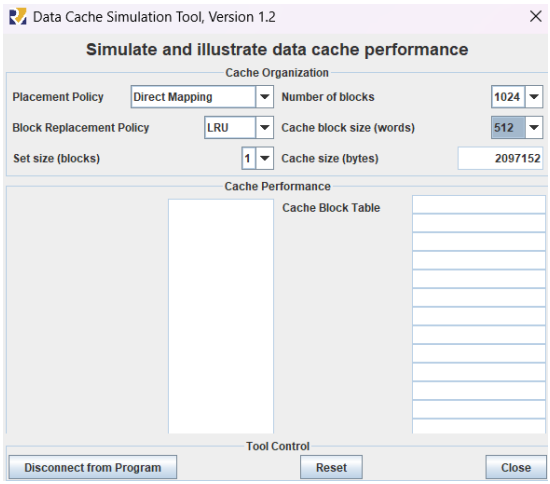

Figura 1.3

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.

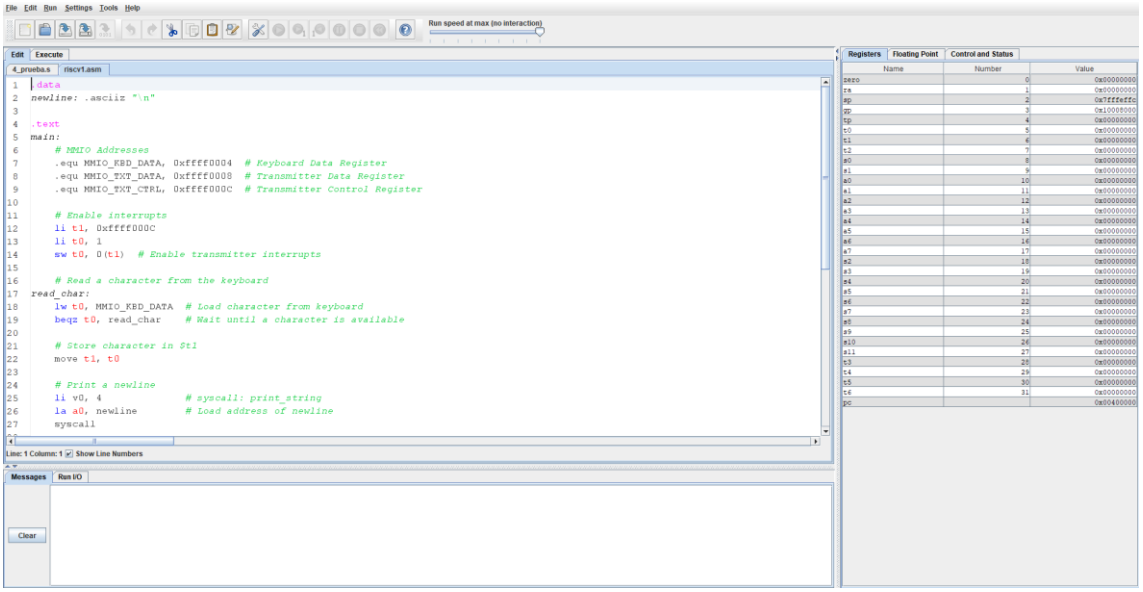

Figura 1.5

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.

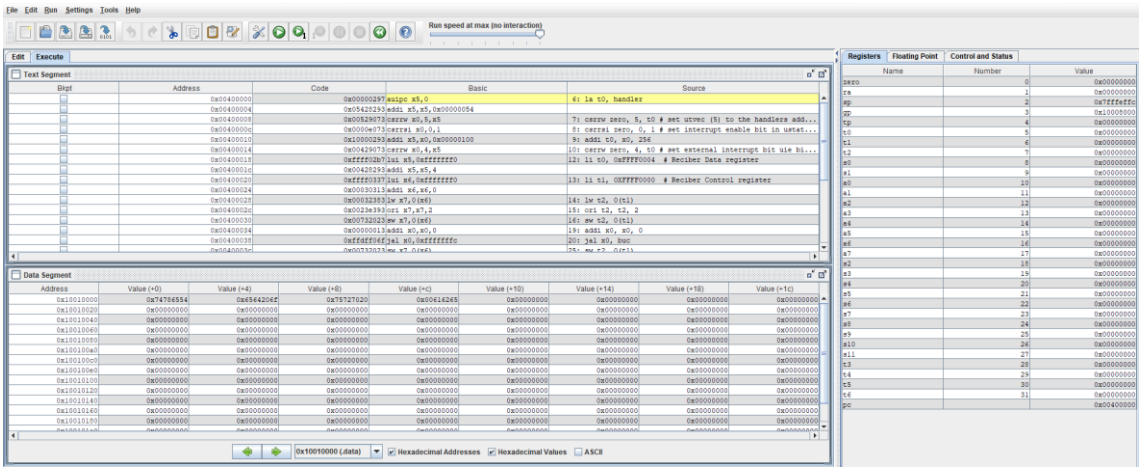

Figura 1.6

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 [8] donde descargamos el archivo de extensión jar.

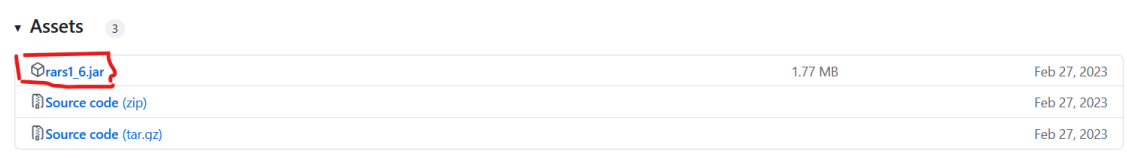

Figura 2.1

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 [9], 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.

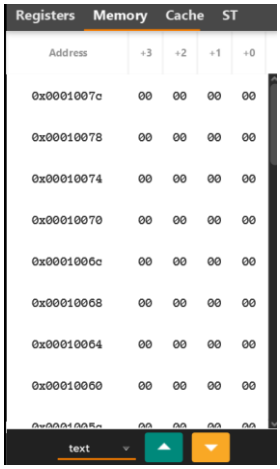

Figura 1.1 (Visualización del contenido en memoria en Jupiter)

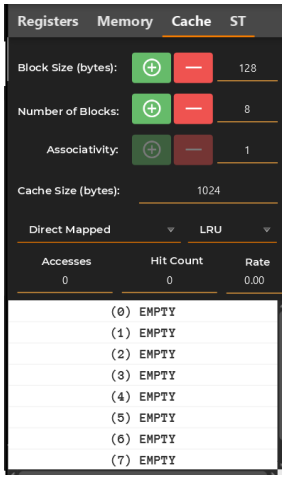

Figura 1.2 (Apartado de memoria caché de Jupiter)

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.

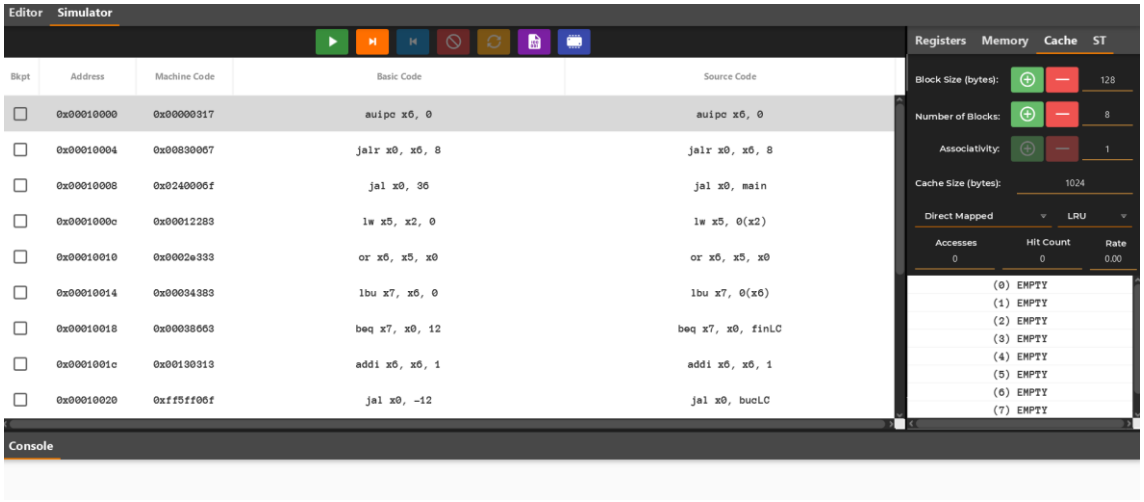

Figura 1.3

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 [9] donde tendremos que instalar el archivo comprimido pertinente al sistema operativo que tengamos disponible.

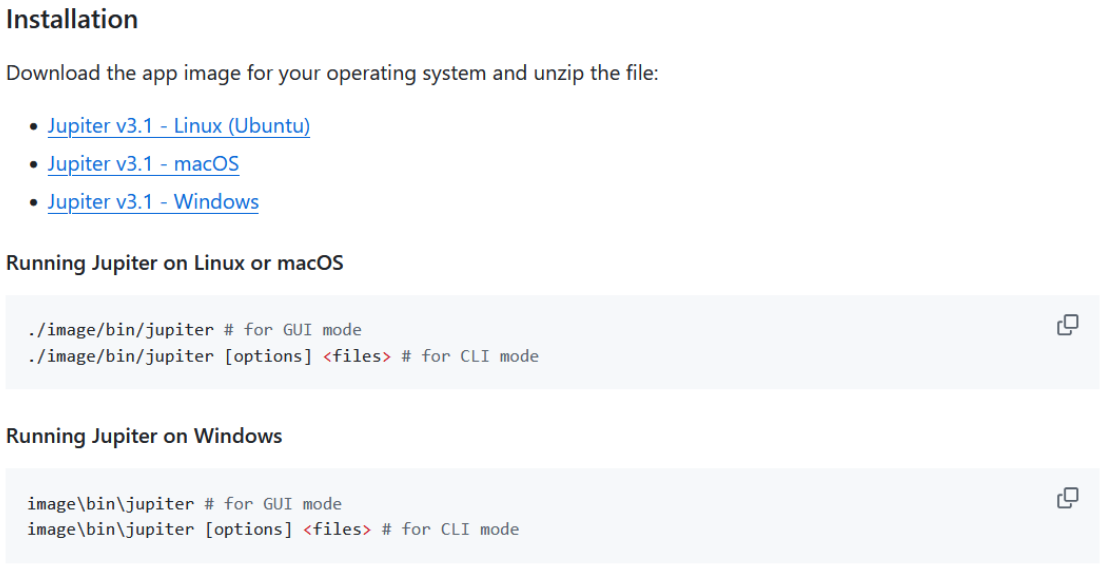

Figura 2.1

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 [10].

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 [11]. Este simulador es un tanto especial puesto que se monta sobre uno de los entornos de programación más usados del sector, Visual Studio Code, aplicación de Microsoft disponible tanto para sistemas operativos Windows como para sistemas operativos Linux, como una extensión de libre desarrollo.

6.4.1.2 Interfaz

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

6.4.1.3 Memorias Caché

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

6.4.1.4 Pipeline

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

6.4.1.5 Entrada Salida

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

6.4.1.6 Trabajo con el simulador

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

6.4.2 Instalación y Ejecución

Este simulador se instala como extensión de la aplicación de Microsoft: Visual Studio Code. Es por esto que el proceso de instalación es prácticamente inmediato siempre que se cuente con la aplicación ya instalada. Aplicación que está disponible independientemente del sistema operativo [12].

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 [13] [14], 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.

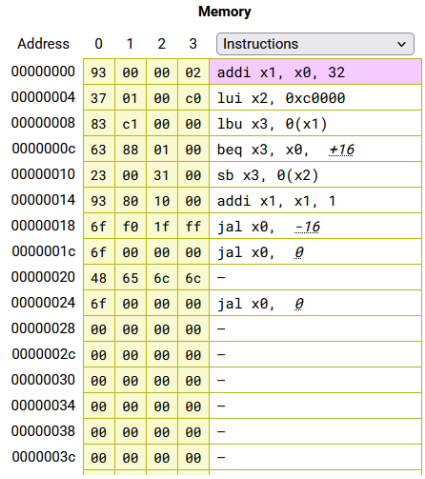

Figura 1.1 (Apartado para el contenido de la memoria principal)

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.

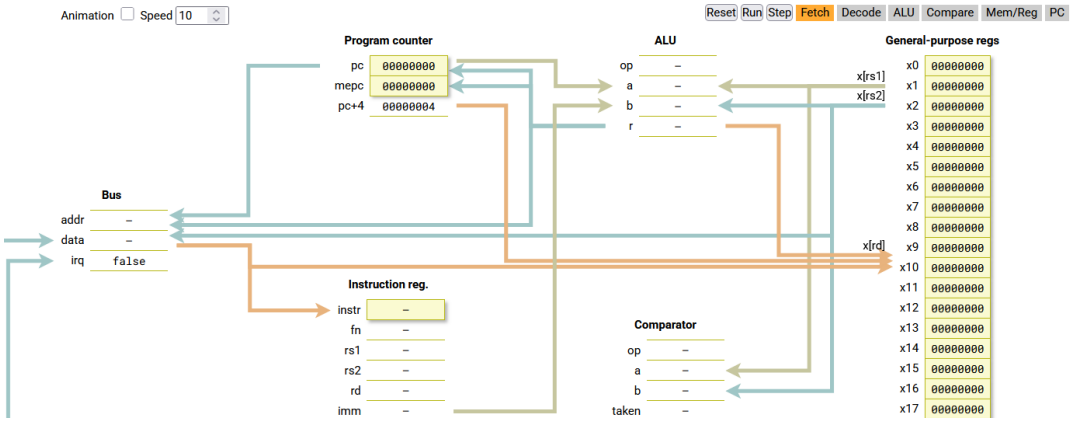

Figura 1.2

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.

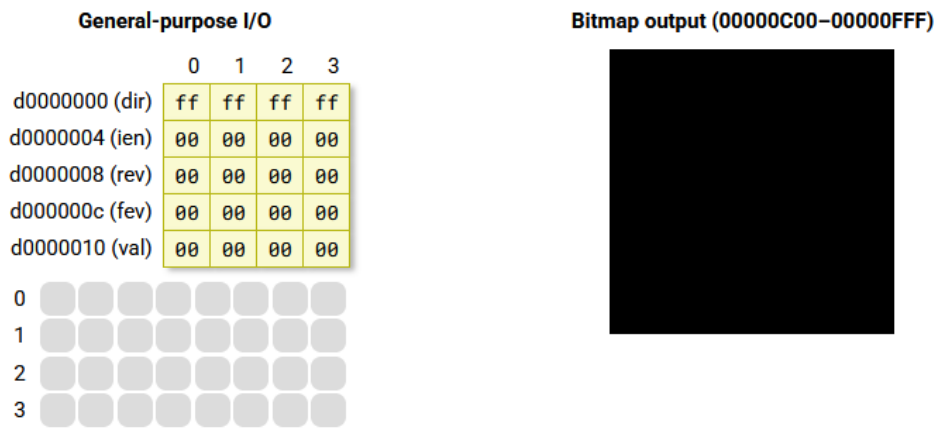

Figura 1.3

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.

```
bruno@MVUbuntu:~/riscv/emulsiV$ npm install
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was created with an old version of npm,
npm WARN old lockfile so supplemental metadata must be fetched from the registry.
npm WARN old lockfile
npm WARN old lockfile This is a one-time fix-up, please be patient...
npm WARN old lockfile
npm WARN deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache i
f you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerfu
l.
npm WARN deprecated rimraf@2.6.3: Rimraf versions prior to v4 are no longer supported
npm WARN deprecated debug@4.2.0: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when u
sed in a Node.js environment. It is recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/iss
ues/797)
npm WARN deprecated debug@4.2.0: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when u
sed in a Node.js environment. It is recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/iss
ues/797)
npm WARN deprecated glob@7.1.6: Glob versions prior to v9 are no longer supported
npm WARN deprecated fontsource-arsenal@3.0.9: Package relocated. Please install and migrate to @fontsource/arsenal.
npm WARN deprecated eslint@7.11.0: This version is no longer supported. Please see https://eslint.org/version-support fo
r other options.
npm WARN deprecated fontsource-roboto@3.0.3: Package relocated. Please install and migrate to @fontsource/roboto.
npm WARN deprecated fontsource-roboto-mono@3.0.3: Package relocated. Please install and migrate to @fontsource/roboto-mo
no.
```

Figura 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”.

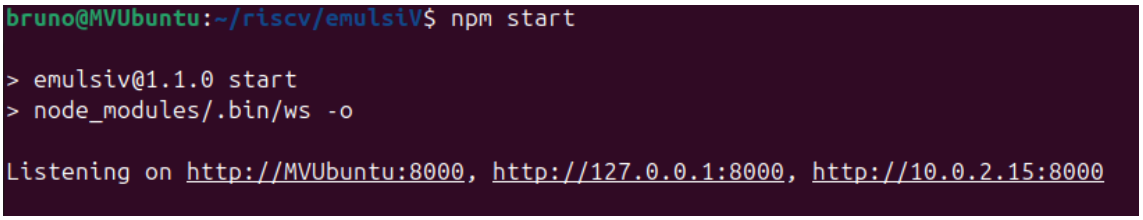

Figura 2.2

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 [15] [16] [17], 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.

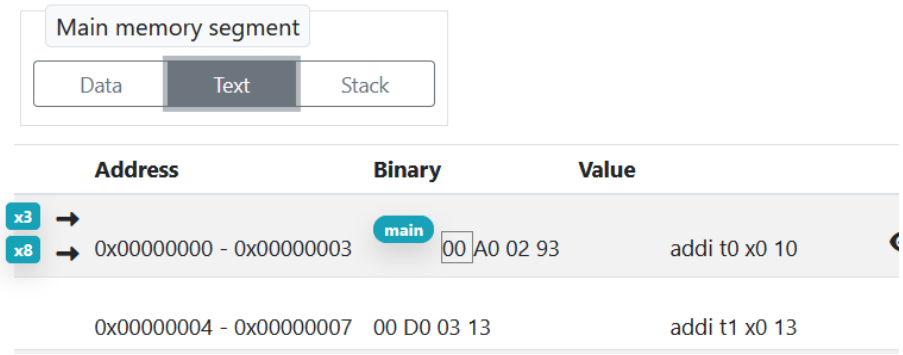

Figura 1.1

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 [18] [19], 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.

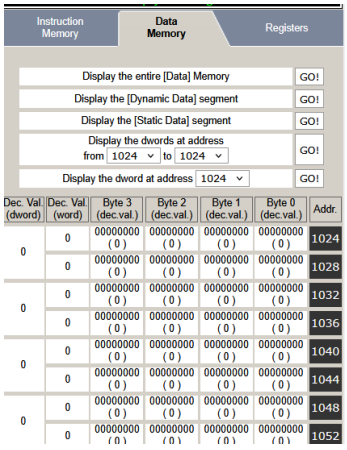

Figura 1.1

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.

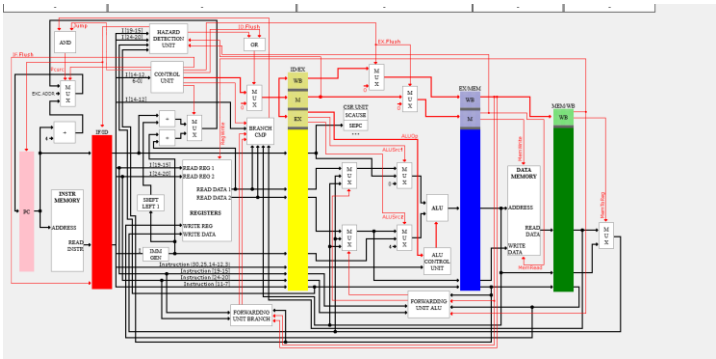

Figura 1.2

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 [20].

6.7.3 Aspectos Destacables y Limitaciones

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

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

depuración se queda muy por detrás. Añadiendo a esto que la interfaz de depuración y ejecución deja mucho que desear.

7 Referencias

[1] RISC-V, «Ratified RISC-V Specifications,» [En línea]. Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications.

[2] Tomshardware, «Nvidia to ship a billion of RISC-V cores in 2024,» [En línea]. Available: https://www.tomshardware.com/pc-components/gpus/nvidia-to-ship-a-billion-of-risc-v-cores-in-2024.

[3] SiFIve, «The History of RISC-V,» [En línea]. Available: https://www.sifive.com/about/risc-v-history.

[4] SiFive, «SiFive Performance P800-Series,» [En línea]. Available: https://www.sifive.com/cores/performance-p870d.

[5] mortbopet, «Repositorio Oficial de RIPES,» [En línea]. Available: https://github.com/mortbopet/Ripes.

[6] Fuse, «FUSE,» [En línea]. Available: https://github.com/AppImage/AppImageKit/wiki/FUSE.

[7] TheThirdOne, «Rars Repository,» [En línea]. Available: https://github.com/TheThirdOne/rars.

[8] TheThirdOne, «RARS Releases,» [En línea]. Available: https://github.com/TheThirdOne/rars/releases/tag/v1.6.

[9] andrescv, «/andrescv/jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter.

[10] «Jupiter,» [En línea]. Available: https://jupitersim.gitbook.io/jupiter/es.

[11] hm-riscv, «/hm-riscv/vscode-riscv-venus,» [En línea]. Available: https://github.com/hm-riscv/vscode-riscv-venus.

[12] «Visual Studio Marketplace RISC-V Venus Simulator,» [En línea]. Available: https://marketplace.visualstudio.com/items?itemName=hm.riscv-venus.

[13] ESEO-Tech, «/ESEO-Tech/emulsiV,» [En línea]. Available: https://github.com/ESEO-Tech/emulsiV.

[14] ESEO-Tech, «/ESEO-Tech/emulsiV/doc,» [En línea]. Available: https://eseo-tech.github.io/emulsiV/doc/.

[15] [En línea]. Available: https://creatorsim.github.io/creator/.

[16] [En línea]. Available: https://github.com/creatorsim/creator.

[17] [En línea]. Available: https://creatorsim.github.io/creator/.

[18] [En línea]. Available: https://github.com/Mariotti94/WebRISC-V.

[19] [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki.

[20] [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation.

[21] mortbopet, «/Ripes/Docs,» [En línea]. Available: https://github.com/mortbopet/Ripes/tree/master/docs.

[22] andrescv, «Repositorio Oficial Jupiter,» [En línea]. Available: https://github.com/andrescv/jupiter.

[23] «Guía Instalación WebRISC-V,» [En línea]. Available: https://github.com/Mariotti94/WebRISC-V/wiki/Local-Installation.