**RARS**

**Índice:**

1

**1 Simulador** 2

1.1 Introducción 2

1.2 Interfaz 2

1.3 Memorias Caché 2

1.4 Pipeline 3

1.5 Entrada Salida 3

1.6 Trabajo con el simulador 4

1.7 Adicionales 4

2

**2 Instalación y Ejecución** 5

3

**3 Complicaciones Encontradas** 5

4

**4** **Referencias** 5

**1 Simulador:**

**1.1 Introducción:**

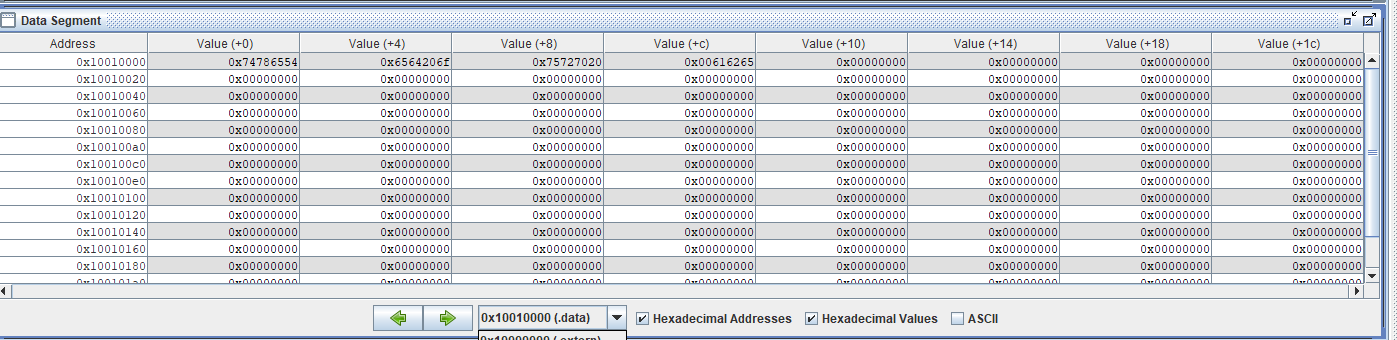
Este documento se dedicará a la exploración del simulador del estándar RISC-V [1], RARS [2], sus posibilidades y sus principales funciones.

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.

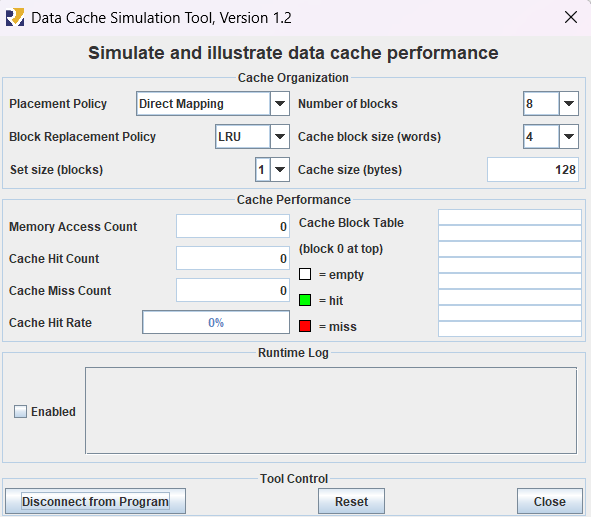
**1.2 Interfaz:**

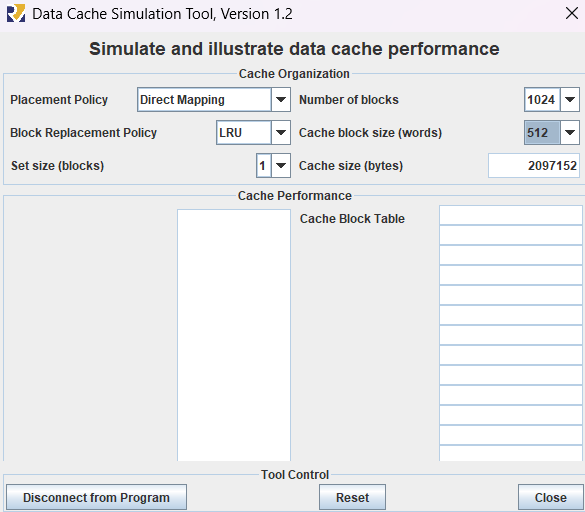
La interfaz de RARS, como se había mencionado con anterioridad es una interfaz gráfica que nos permite visualizar el flujo del programa resaltando las instrucciones que se están ejecutando en el momento.

**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

**1.4 Pipeline:**

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

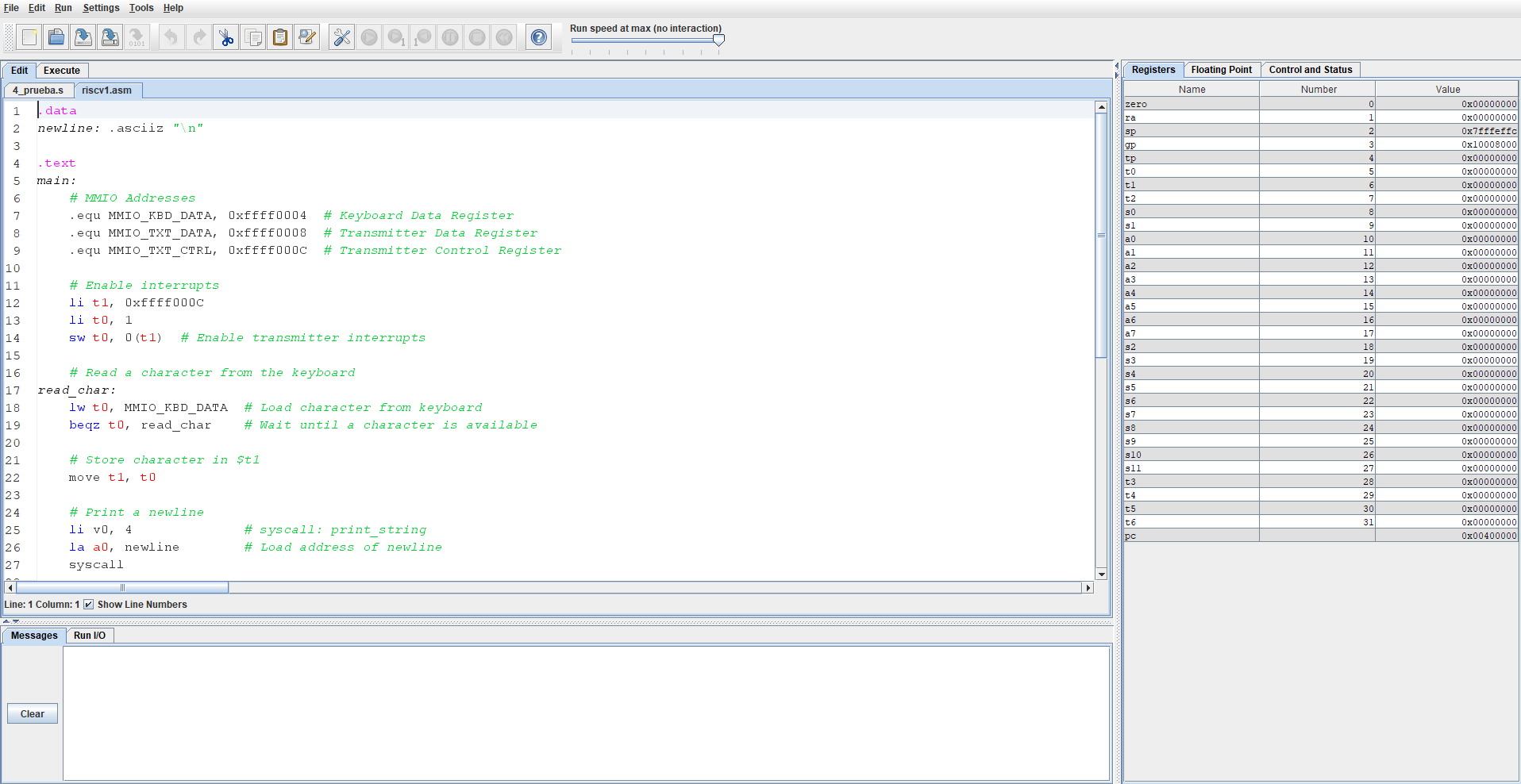
**1.5 Entrada Salida:**

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.

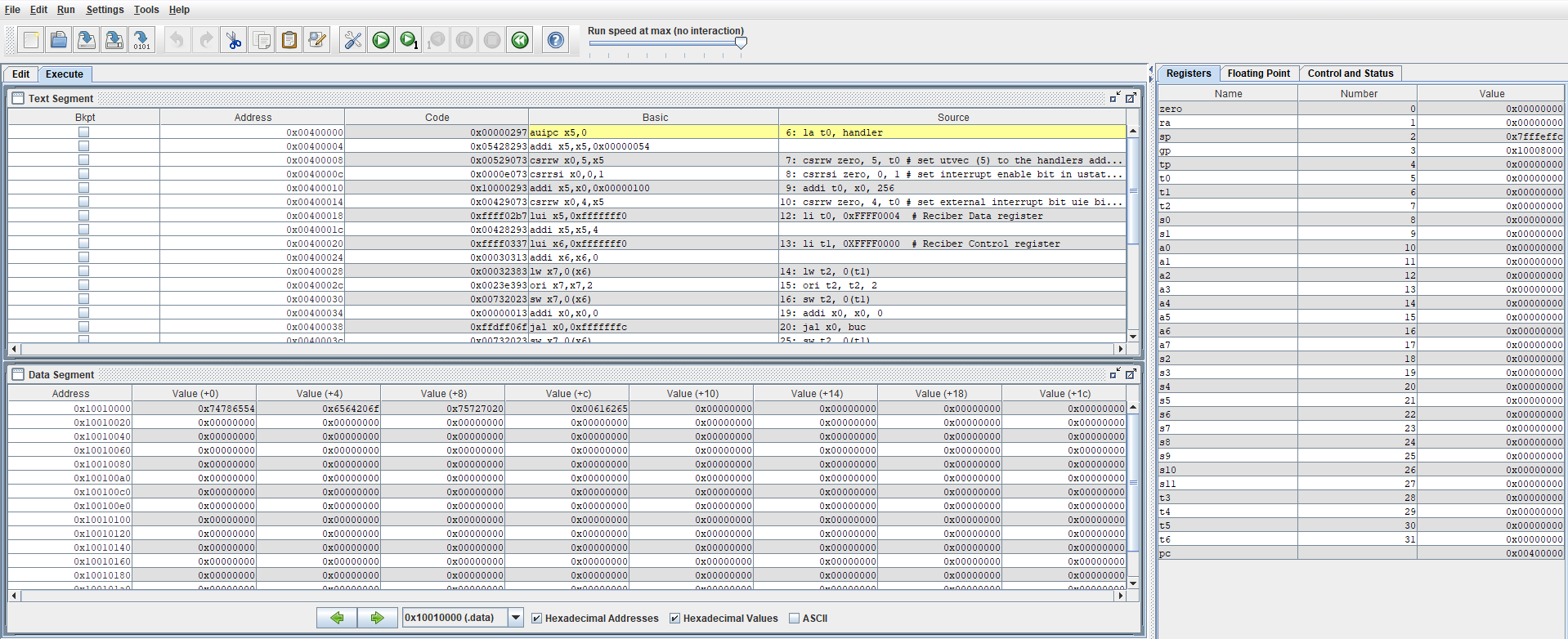
Sin embargo, incluso cuando cuenta con algunos periféricos implementados con los que se puede jugar con la entrada salida y las interrupciones, las instrucciones sobre como estos funcionan no están muy claras, y adicionalmente tienen algunas fallas y aspectos a mejorar en el funcionamiento de los mismos.

**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

**1.7 Adicionales:**

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.

**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 [2] 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.

**3 Complicaciones Encontradas:**

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.

# 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] | TheThirdOne, «RARS Releases,» [En línea]. Available: https://github.com/TheThirdOne/rars/releases/tag/v1.6. |
| [3] | TheThirdOne, «Rars Repository,» [En línea]. Available: https://github.com/TheThirdOne/rars. |