CREATOR: Simulador didáctico y genérico para la programación en ensamblador

Diego Camarmas-Alonso[1] Félix García-Carballeira[2] Elías Del-Pozo-Puñal[3] y Alejandro Calderón Mateos[4]

Resumen— **En este artículo se presenta CREATOR, un simulador para la programación en ensamblador desarrollado por el grupo ARCOS de la UC3M [1]. Este simulador permite definir la sintaxis y el funcionamiento de cualquier juego de instrucciones así como el convenio de paso de parámetros utilizado. Una vez definido cada juego de instrucciones en particular (MIPS32, ARM, RISC-V, etc.), los estudiantes pueden utilizar el simulador para editar, compilar, ejecutar y depurar programas escritos en ensamblador. El simulador permite también comprobar que los programas escritos cumplen con el convenio de paso de parámetros que se ha definido para cada ensamblador. Todo ello, mejora la enseñanza y uso del lenguaje ensamblador presente en distintas asignaturas como Estructura de Computadores o Arquitectura de Computadores.**

CREATOR es un nuevo simulador altamente intuitivo y portable, que se ejecuta directamente en un navegador web, sin necesidad de ningún tipo de instalación previa.

Este simulador inicialmente dispone del juego de instrucciones MIPS32 y RISC-V (32IMF), pero permite, desde el propio simulador, editar y definir el juego de instrucciones (instrucciones, formato, registros, convenio de paso de parámetros, etc.) de cualquier computador. El simulador dispone de la capacidad para crear bibliotecas de subrutinas que pueden cargarse y enlazarse con otros programas escritos en el simulador. Ello permite construir laboratorios de prácticas más adaptados a los objetivos docentes deseados.

La experiencia de su uso ha sido muy positiva en los cursos 2019/2020 y 2020/2021 para los estudiantes, especialmente en el entorno de enseñanza vivido por el COVID-19.

Palabras clave— **Estructura de Computadores, Arquitectura de Computadores, Programación en ensamblador, MIPS32, RISC-V32.**

I. Introducción

El aprendizaje de lenguaje ensamblador no solo ayuda a entender mejor el funcionamiento de un computador, también es de ayuda a la hora de optimizar código. Por ejemplo, para escribir una porción de código que debe ejecutar de la forma más rápida posible o, también, para ajustar el código ensamblador generado por el compilador.

Habitualmente, en la docencia de un lenguaje ensamblador se utiliza un simulador específico que ejecuta como aplicación de PC (por ejemplo, QtSPIM, SPIM, etc.). Por otra parte, usar una CPU real suele complicar el entorno de trabajo (por ello, se usa un simulador) y, hasta donde los autores de este trabajo conocen, no hay un simulador que permita trabajar con distintos juegos de instrucciones.

Por un lado, si se precisa aprender dos o más lenguajes en ensamblador a la vez (por ejemplo, MIPS [2] y RISC-V [3]), o hacer la transición de un lenguaje de ensamblador a otro (por ejemplo, pasar de MIPS a RISC-V) entonces, se precisa trabajar con varios simuladores distintos (que conlleva un tiempo de adaptación a cada simulador). Ello eleva complejidad en el proceso de aprendizaje del lenguaje ensamblador.

Por otro lado, disponer de un simulador a través de un navegador web facilita su ejecución en cualquier plataforma y permite una mayor movilidad en el aprendizaje (uso de móvil, tablets, etc. para continuar el aprendizaje fuera de clase). La mayor parte de las herramientas están pensadas para ejecutarse en un PC con un conjunto limitado de sistemas operativos, lo que dificulta la adaptación a nuevos entornos y necesidades que una situación con la pandemia vivida precisa.

Para ayudar a solucionar estos problemas proponemos CREATOR [1] (didaCtic geneRic assEmbly progrAmming simulaTOR), que es: simple, intuitivo, genérico, multiplataforma y en el que se puede comprender el funcionamiento de un computador a bajo nivel gracias a la creación y ejecución de programas en diferentes lenguajes ensamblador (inicialmente MIPS32 y $RISCV32_{IMF}$) y a la gran libertad de la que disponen los estudiantes y profesores para editar los principales aspectos del juego de instrucciones y arquitectura sobre la que se está trabajando.

El resto del documento se estructura de la siguiente forma: la Sección II describe el Estado del Arte; la Sección III presenta el Simulador CREATOR y sus distintas visiones. Por último, la Sección IV presenta las principales Conclusiones y Trabajos Futuros.

II. Estado del arte

Para el aprendizaje de lenguaje ensamblador habitualmente se utilizan los simuladores SPIM, MARS y WebMIPS [4], [5], [6].

SPIM [7] permite ejecutar (y depurar) programas ensamblador sobre una arquitectura MIPS de 32 bits. Este simulador ejecuta como aplicación de PC y está disponible para Microsoft Windows, Mac OS X y Linux. La interfaz de este simulador está basada en la terminal del sistema operativo. No obstante, se ha desarrollado una variante de SPIM llamada QtS-

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: dcamarma@inf.uc3m.es.

[2]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: fgcarbal@inf.uc3m.es.

[3]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: edelpozo@inf.uc3m.es.

[4]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: acaldero@inf.uc3m.es.

[1]Disponible en https://creatorsim.github.io/

Tabla I: Comparación de CREATOR con los simuladores del estado del arte.

Simulador	**SPIM**	**QtSPIM**	**MARS**	**RARS**	**WebMIPS**	**CREATOR**
Multiplataforma	C/C++	C/C++	Java	Java	HTML5	HTML5
Juego de instrucciones personalizable						✓
Arquitectura editable						✓
Convenio de parámetros						✓
Visualización de marco pila						✓
Línea de mandatos	✓		✓	✓		✓
Interfaz gráfica		✓	✓	✓	✓	✓

PIM que ofrece una interfaz de ventanas que lo hace más intuitivo e interactivo. Desafortunadamente, SPIM y QtSPIM solo permiten ejecutar el juego de instrucciones de MIPS32 (con la extensión de la instrucción *syscall*), por lo que no se puede modificar el juego de instrucciones existente, ni añadir nuevas arquitecturas como RISC-V. Además, su diseño no está pensado para dispositivos móviles y no integra un editor de código.

MARS [8] es un simulador educativo que permite simular la arquitectura MIPS de 32 bits, al igual que el simulador SPIM. Este simulador destaca por estar desarrollado en Java, disponer de interfaz gráfica con un editor de código ensamblador integrado y una utilidad que permite al usuario acceder al hardware de la arquitectura desde el ensamblador. Sin embargo, al igual que sucede con el simulador anterior, no permite editar el juego de instrucciones cargado, añadir nuevas arquitecturas al simulador, ni dispone soporte para dispositivos móviles.

RARS [9] es un simulador que permite ejecutar programas ensamblador en RISC-V IMFDN. Este simulador está desarrollado sobre la base de MARS, pero no integrado con MARS, por lo que tiene parecidas funcionalidades y limitaciones que MARS.

WebMIPS [10] es un simulador web que permite ejecutar programas ensamblador para la arquitectura MIPS de 32 bits. A diferencia de los simuladores descritos anteriormente, este simulador permite ejecutar con segmentación (*pipeline*) de cinco etapas. Desafortunadamente, este simulador solo implementa un subconjunto del juego de instrucciones de MIPS y tampoco permite editar dicho juego para añadir nuevas instrucciones o definir un juego de instrucciones de otra arquitectura diferente a MIPS.

Por tanto, como podemos observar en la Tabla I, CREATOR reúne las características de las diversas herramientas comentadas anteriormente, facilitando el aprendizaje de forma iterativa en un único simulador.

III. CREATOR

La forma más fácil de empezar a trabajar con CREATOR es indicar un enlace que permite cargar en el navegador Web la herramienta y además cargar un ejemplo listo para ejecutar. CREATOR dispone de una lista de ejemplos en el proyecto GitHUB [2] que pueden cargarse desde un enlace.

Por ejemplo, para mostrar cómo funciona un *factorial* de forma recursiva es posible usar el enlace: https://creatorsim.github.io/creator/?example_set=default_rv&example=e12.

La Tabla II muestra la lista de ejemplos disponibles de RISC-V. También, es posible definir un conjunto propio de ejemplos modificando en el enlace el identificador del conjunto (*default_rv* en el ejemplo anterior) y el identificador del ejemplo (*e12* en el anterior ejemplo).

Además de usar los ejemplos básicos, se puede trabajar con CREATOR seleccionando alguna de las arquitecturas disponibles (MIPS, RISC-V, etc.) y editando en la pantalla de ensamblador el código ensamblador del programa a probar. Una vez editado y compilado sin fallos es posible pasar a la pantalla de ejecución donde se podrá ejecutar el programa implementado.

Cabe destacar que CREATOR incorpora un sistema que realiza una copia de seguridad del código ensamblador introducido en el editor cada vez que se compila este. El código es almacenado en la caché del navegador y cada vez que se inicia el simulador se comprueba si se encuentra alguna copia disponible para ser cargada, si existe, se le da la posibilidad al estudiante de cargarla para seguir desarrollando el código (si se utiliza modo privado en el navegador está no se conservará). Este sistema ayuda a que los estudiantes no pierdan todo su trabajo de forma accidental por un corte de luz o un problema en el dispositivo que estén utilizando.

Además de poder usar las arquitecturas existentes, es posible editar dichas arquitecturas (instrucciones, segmentos de memoria, banco de registros, etc.) o incluso definir una nueva arquitectura. Tener en CREATOR la posibilidad de trabajar con múltiples arquitecturas facilita una adaptación más rápida y cómoda a escenarios muy comunes: migrar de MIPS a RISC-V, poder comparar dos códigos en ensamblador, etc.

Este simulador también dispone de una versión en línea de mandatos que es de gran utilidad para realizar pruebas (y correcciones) de la forma más automatizada posible. Permite la ejecución y la visualiza-

[2]Disponible en https://github.com/creatorsim/creator#examples-included-in-creator

Tabla II: Conjunto de ejemplos ofrecidos con RISC-V en CREATOR

Description	Link
Data Storage	https://creatorsim.github.io/creator/?example_set=default_rv&example=e1
ALU operations	https://creatorsim.github.io/creator/?example_set=default_rv&example=e2
Store/Load in Memory	https://creatorsim.github.io/creator/?example_set=default_rv&example=e3
FPU operations	https://creatorsim.github.io/creator/?example_set=default_rv&example=e4
Loop	https://creatorsim.github.io/creator/?example_set=default_rv&example=e5
Branch	https://creatorsim.github.io/creator/?example_set=default_rv&example=e6
Loop + Memory	https://creatorsim.github.io/creator/?example_set=default_rv&example=e7
Copy of matrices	https://creatorsim.github.io/creator/?example_set=default_rv&example=e8
I/O Syscalls	https://creatorsim.github.io/creator/?example_set=default_rv&example=e9
I/O Syscalls + Strings	https://creatorsim.github.io/creator/?example_set=default_rv&example=e10
Subrutines	https://creatorsim.github.io/creator/?example_set=default_rv&example=e11
Factorial	https://creatorsim.github.io/creator/?example_set=default_rv&example=e12

ción del valor de los registros, posiciones de memoria y contenido de dispositivos que sean distintos a su valor inicial (cero tras un reset) al finalizar la ejecución. De esta forma este conjunto de valores que forma el estado tras la ejecución se puede comparar con uno esperado, como se puede ver en los Listados 1 y 2 y comprobar si el resultado final coincide con el caso correcto, y en caso de que difiera ver donde ocurre. Para guardar el estado correcto de una ejecución en el archivo *salida.txt* se usará:

Listado 1: Ejemplo con CREATOR para guardar el estado

```
# ./creator.sh \
    -a ./architecture/MIPS-32-like.json \
    -s ./examples/MIPS/example2.txt \
    -o min > salida.txt
```

Por otro lado, para ejecutar el código escrito en *example2.txt*, usando la arquitectura *MIPS-32-like.json* y comparar con el estado contenido en *salida.txt* se usará:

Listado 2: Ejemplo usando CREATOR para comprobar una ejecución

```
# ./creator.sh \
    -a ./architecture/MIPS-32-like.json \
    -s ./examples/MIPS/example2.txt \
    -o min \
    -r salida.txt
```

La accesibilidad es muy importante en una herramienta educativa, y en CREATOR se ha trabajado en mejorar la accesibilidad tanto en la versión en Web como en la versión en línea de mandatos. En la versión web CREATOR 2.1.x ha pasado el estándar WCAG 2.0 y nivel AAA [11]. Por ejemplo, desde el modal de configuración de CREATOR se puede cambiar la velocidad de ejecución y el tiempo que se muestran las notificaciones. Y desde el el botón de información se puede acceder al historial de notificaciones. Ambas facilitan la lectura y comprensión para las personas que lo necesiten. En la versión de línea de mandatos la salida se ha formateado de forma que permite a los estudiantes con dificultades de visión utilizar lectores de pantalla.

A. Definición del juego de instrucciones

La pantalla de arquitectura (Figura 1) permite a los profesores (o cualquier persona que quiera editar o crear una arquitectura) definir la disposición (y tamaño) de los segmentos de memoria, definir registros y bancos de registro, instrucciones, pseudoinstrucciones y directivas. Para evitar que un usuario novel accidentalmente modifique la arquitectura seleccionada, se ha añadido un botón de modo avanzado que habilita estas opciones.

Podemos destacar de la edición de la arquitectura:

- En la edición de registros, es posible indicar varios nombres para un mismo registro (alias) que es usado, por ejemplo, en RISCV32. Para los registros de doble precisión de coma flotante, es posible juntar de dos en dos registros de simple precisión. Además, permite indicar una serie de propiedades a los registros:
  - Indicar si un registro tiene permisos de lectura, escritura o ambos.
  - Si es un puntero especial a una zona de memoria para mostrarlo en el panel de memoria en la pantalla de ejecución.
  - Definir aspectos del convenio de paso de parámetros como que registros han de preservarse entre llamadas.
- En la edición de las instrucciones y pseudoinstrucciones (Figura 1) es posible indicar:
  - El formato de la instrucción (número de palabras, número de campos, etc.).
  - Propiedades de la instrucción (inicio de subrutina y final de subrutina).
  - En cada campo se indica su codificación (tipo, bit de inicio, bit de fin, etc.)
  - La definición de la funcionalidad de la instrucción (se define en un lenguaje de alto nivel que es JavaScript, pudiendo añadir llamadas a funciones del API del simulador para el acceso a componentes hardware, definición del convenio de paso de parámetros, entre otros). En el caso de las pseudoinstrucciones, en la descripción se indica la secuencia de instrucciones a las que equivale la pseudoinstrucción.
- En las directivas se pueden definir varias directivas para una misma acción, así como indicar el tamaño en bytes de las directivas que representan tipos de datos.

Para ayudar a solventar los errores de edición de arquitectura, se han añadido diversas facilidades como, por ejemplo:

- Validación de formularios con colores (indicación de campos obligatorios vacíos)

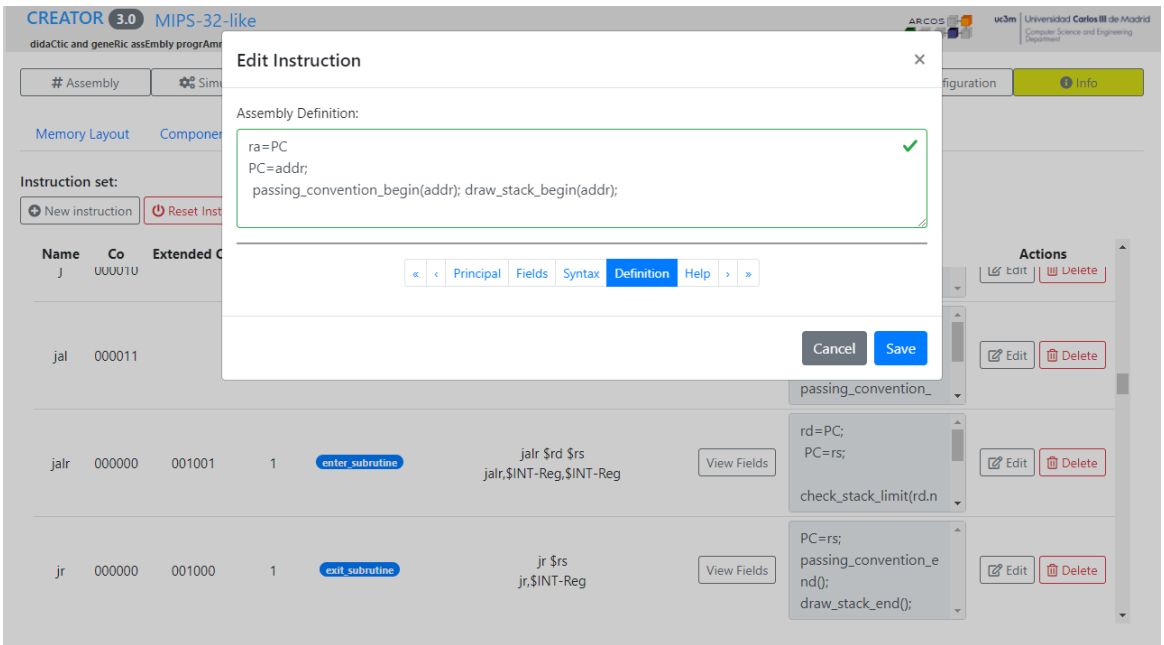

Fig. 1: Definición de arquitectura en CREATOR.

- Comprobación del tipo correcto de datos en los campos (numéricos, decimales, binarios, etc.)
- En el caso de las pseudoinstrucciones se revisa que las instrucciones usadas en su definición existan en el juego de instrucciones.

Para realizar todas las acciones descritas en las líneas anteriores la pantalla de definición de arquitectura (Figura 1) se divide en una barra de botones en la parte superior que permite cambiar de pantalla, guardar la arquitectura actual en formato JSON y activar y desactivar el modo avanzado. Y, a continuación, se muestran las tablas y formularios de edición de los diferentes componentes que conforman la arquitectura (registros, instrucciones, pseudoinstrucciones, etc.).

B. Desarrollo de programas ensamblador

Una vez definido el juego de instrucciones del ensamblador deseado, se pueden realizar programas escritos en este ensamblador.

Para ello se dispone de la pantalla de ensamblador (Figura 2) que permite a los estudiantes implementar sus propios programas ensamblador o cargar ejemplos que ya están predefinidos en el simulador.

Una vez los estudiantes han implementado su programa o han cargado un ejemplo pueden compilar el código ensamblador para cargarlo en memoria y proceder posteriormente a su ejecución, o en caso de que exista algún error en el código ver cuál es y subsanarlo.

CREATOR informa de los errores existentes en el código a los estudiantes de manera que puedan corregirlos para que el programa pueda compilar.

Además, mejora la descripción de los errores al indicar el fragmento de código donde se ha detectado el fallo y al añadir una breve explicación del error encontrado, que ayuda al estudiante en su resolución.

Este simulador es capaz de informar de hasta 24 errores distintos. Ofreciendo en todos ellos una descripción del error lo más informativa posible.

En la Figura 3 se puede ver un ejemplo de un error sintáctico con una etiqueta que no existe en RISC-V.

Además, para facilitar a los estudiantes la implementación de sus programas en esta pantalla está disponible una ayuda en línea que describe las instrucciones de la arquitectura seleccionada (ver Figura 2).

Para realizar todas las acciones descritas anteriormente la pantalla ensamblador (Figura 2) se divide en una barra de botones en la parte superior que permite:

- Navegar a otras pantallas (arquitectura y simulador).
- Interactuar con el editor de texto de manera que se pueda crear un nuevo programa, cargar uno existente en el dispositivo, guardar el contenido del editor en el dispositivo y cargar un ejemplo de los que ofrece el simulador.
- Compilar el código implementado en el editor.
- Crear una biblioteca que contiene subrutinas que pueden ser utilizadas por otros programas. También permite enlazar una biblioteca que ya haya sido creada anteriormente con el programa que se está implementando en el editor. Por último, desenlazar la biblioteca que no se quiera utilizar.

A continuación, se dispone de un editor de texto integrado en el simulador que se ha configurado especialmente para facilitar la edición de código ensamblador. No solo usa colores y numeración de línea, sino también se han habilitado una serie de atajos de teclado típicos de la mayoría de editores de código fuente (VSCode, Sublime Text, CLion, etc.)

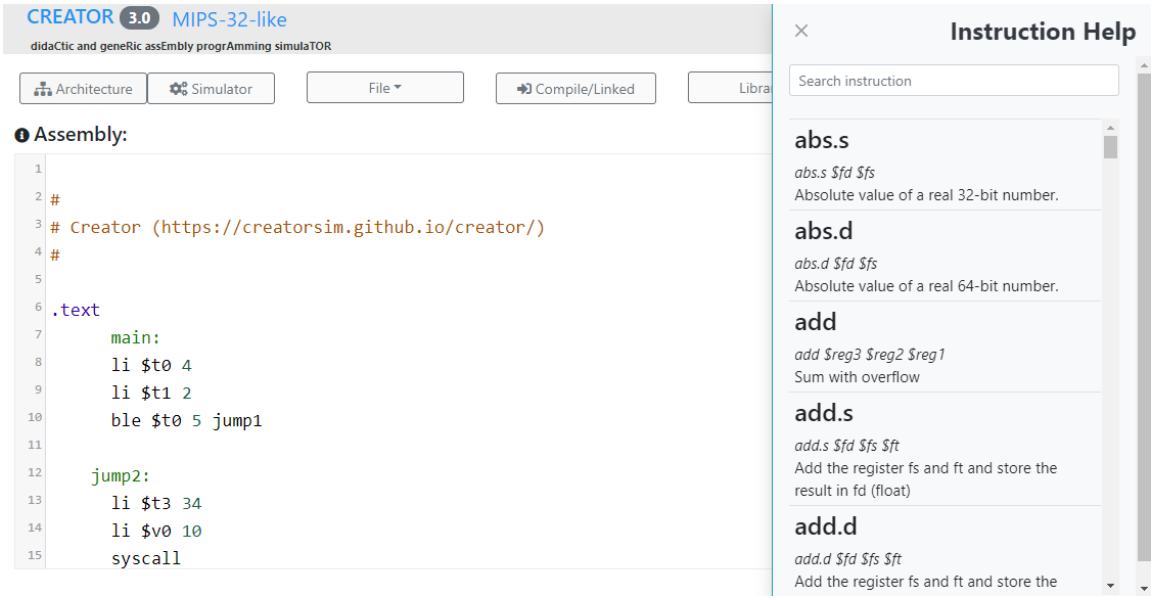

Fig. 2: Ejemplo de edición con ayuda en línea de instrucciones.

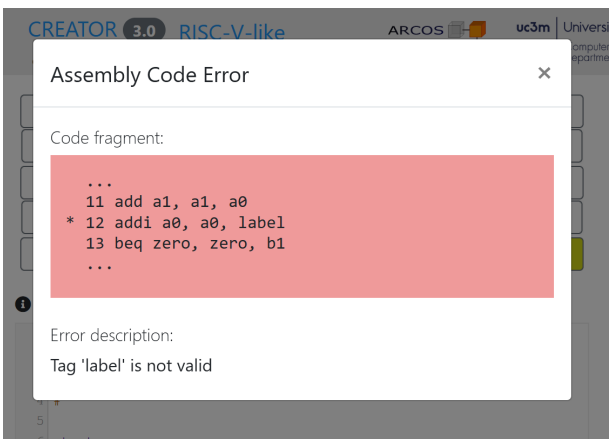

Fig. 3: Ejemplo de mensaje de error.

C. *Ejecución de programas*

La pantalla de ejecución permite a los estudiantes ejecutar los programas ensamblador que hayan implementado o alguno de los ejemplos existentes en CREATOR. Durante la ejecución del programa pueden visualizar el estado de la ejecución, el valor almacenado en los registros y en la memoria, así como, estadísticas de ejecución.

La pantalla (Figura 4) se divide en una barra de botones en la parte superior, la lista de instrucciones a ejecutar en la parte izquierda, un panel de detalles en la parte derecha (con la información de registros, memoria y estadísticas) y, en la parte inferior, una pantalla y un teclado para interactuar con las llamadas al sistema de E/S.

La barra de botones permite (Figura 4):

a) Navegar a otras pantallas (arquitectura y ensamblador).
b) Reiniciar la ejecución del programa actual, ejecutar el programa instrucción a instrucción (también se puede realizar esta acción con un atajo de teclado) y ejecutar el programa hasta su finalización o un punto de ruptura.
c) Cargar un ejemplo existente en el simulador para la arquitectura seleccionada.
d) Abrir una calculadora con representación IEEE 754 que permite transformar y visualizar un número en formato decimal, hexadecimal y binario.

La lista de instrucciones a ejecutar permite a los estudiantes ver si existe un punto de ruptura en la ejecución del programa (*Break*), la dirección de memoria del segmento de texto donde se almacena cada una de las instrucciones del programa (*Address*), las etiquetas asociadas a determinadas direcciones de memoria (*Label*), las instrucciones o pseudoinstrucciones implementadas por el usuario (*User Instruction*) y las instrucciones que se han compilado y va a ejecutar el simulador (*Loaded Instruction*).

Estos campos permiten a los profesores enseñar a los estudiantes cómo una etiqueta definida en el código ensamblador está asociada a una dirección concreta de memoria o cómo una pseudoinstrucción se transforma durante la compilación en varias instrucciones que permiten realizar su funcionalidad. Además, la posibilidad de definir un punto de ruptura durante la ejecución del código permite a los estudiantes depurar el código ensamblador con una mayor facilidad y rapidez.

Como se puede ver en la Figura 4, durante la ejecución de un programa ensamblador se señalan dos instrucciones, la última instrucción ejecutada (azul) y la siguiente instrucción a ejecutar (verde), que será la indicada por el registro PC. Destacar estas dos instrucciones entre todas las mostradas en la lista facilita a los estudiantes comprender el funcionamiento de las instrucciones que realizan saltos o llamadas a subrutinas ya que permite ver el origen y destino del salto al mismo tiempo.

El panel de detalles permite visualizar los valores

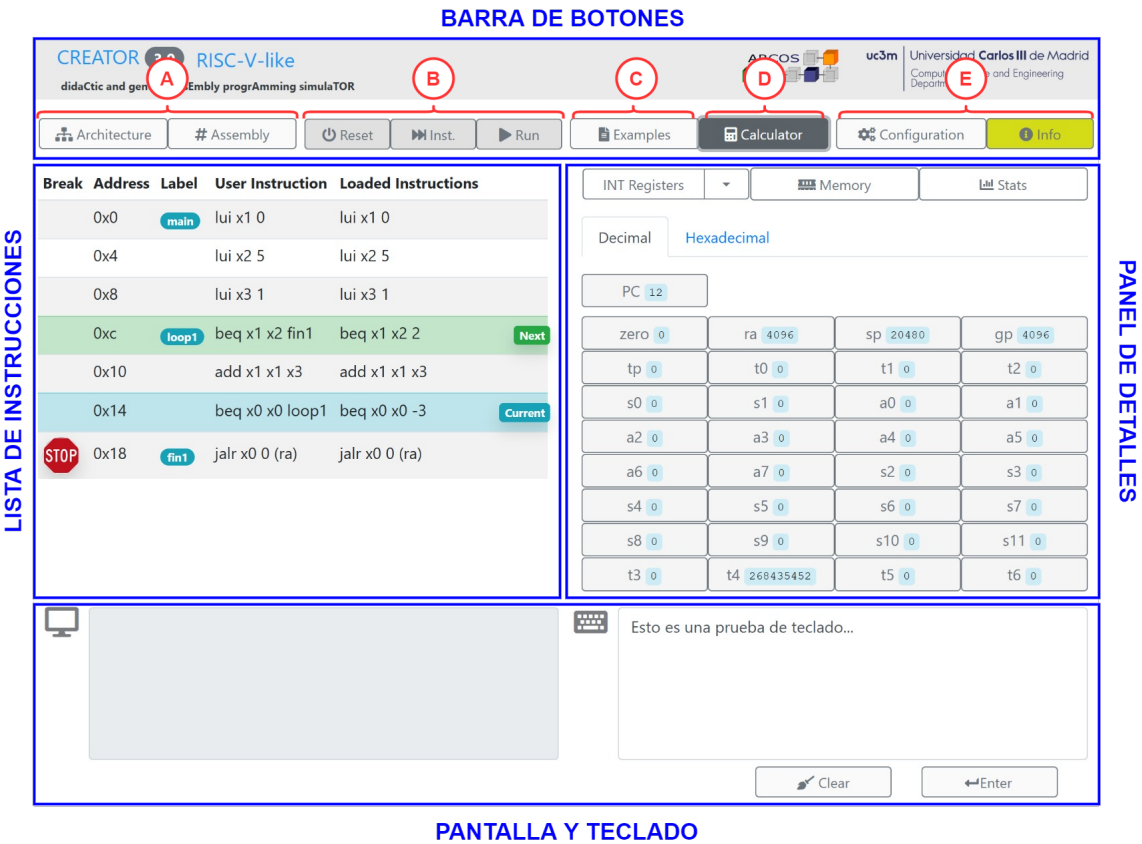

Fig. 4: Ejemplo de ejecución con un bucle y un punto de ruptura.

almacenados en el banco de registros entero y en el de coma flotante de simple (32 bits) y doble precisión (64 bits). Además, estos valores se pueden representar tanto en decimal como en hexadecimal, lo que permite a los estudiantes visualizar los valores en el formato que les sea más sencillo dependiendo de las necesidades de cada instante de la ejecución.

Asimismo, en este panel también se pueden visualizar los valores almacenados en los tres segmentos de memoria (datos, texto y pila). Como se puede ver en la Figura 5, para cada palabra de memoria se indican las direcciones que ocupa esta en memoria, el valor que almacena en formato hexadecimal y, por último, este mismo valor en representación decimal o en caracteres. Para la representación del valor se utilizará el tipo de dato que se indicó en el segmento de datos (.data) del código ensamblador o en el caso de que sea una reserva de espacio de memoria (.space) la representación que el usuario indique.

Mostrar el valor almacenado en memoria utilizando el formato decimal o caracteres permite a los estudiantes saber fácilmente si el programa desarrollado funciona correctamente o no puesto que estas representaciones habitualmente son más legibles y familiares para ellos que el formato hexadecimal utilizado en memoria.

Además, durante la ejecución de un programa ensamblador se pueden detectar errores de ejecución como puede ser la escritura de un dato en el segmento de texto de la memoria o la falta de permisos para leer o escribir en un registro.

C.1 Entender mejor el uso de la pila

En la Figura 5 se muestra el panel de detalles del segmento de memoria de pila. Como se puede observar, se muestra un listado con las direcciones en hexadecimal, el contenido asociado en binario así como el valor usado de origen. En ese listado se marca dónde apuntan los registros puntero. Cuando se definen los registros de la arquitectura se puede indicar cuáles son punteros a zona de memoria como propiedades. En esta Figura 5 están *sp* (*stack pointer*) y *fp* (*frame pointer*). Para ayudar a distinguir los valores guardados en pila por el llamante y por el llamado, se muestran estos con colores azul y verde, respectivamente. En la parte inferior hay una leyenda que resume el estado de la pila de forma que a la derecha se encuentra la parte superior de la pila identificada con *System Stack*, a continuación, están el número de llamadas intermedias a subrutina (*... 1*), luego el último llamante (*Caller: factorial*) y el último llamado (*Callee: factorial*). Por último, a la izquierda se muestra la zona libre de pila (*Free Stack*).

Toda esta información ayuda a las personas que usan el simulador a entender mejor el uso de la pila, que es muy importante tanto para entender las llamadas recursivas como para entender que las variables locales se guardan en pila (y desaparecen tras terminar la llamada, por lo que no se puede devolver una referencia a las mismas).

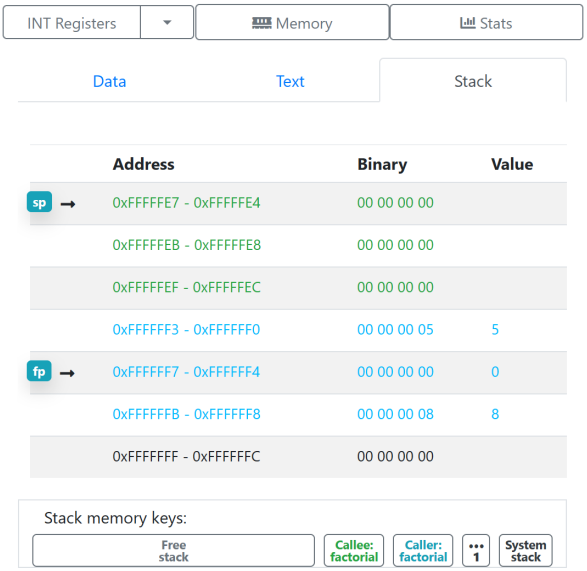

Fig. 5: Ejemplo de segmento de pila con memoria reservada por la función llamada y la llamante.

C.2 Comprobación del convenio de paso de parámetros

Explicar y entender el convenio de paso de parámetros es siempre un reto. A la hora de integrar una rutina en ensamblador con un código en lenguaje de alto nivel es necesario entender el convenio que las herramientas de desarrollo usadas (compilador principalmente) está usando.

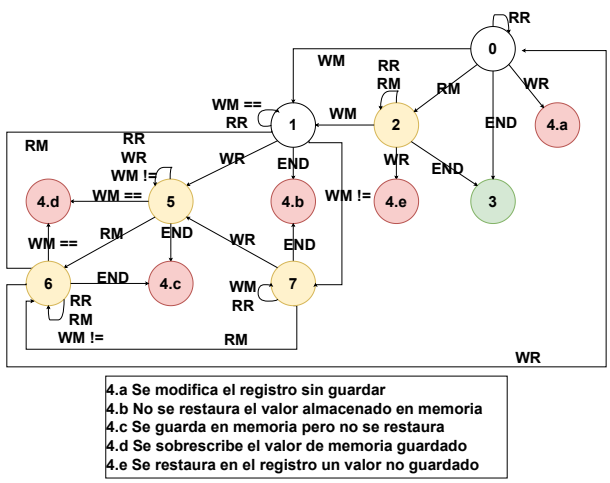

Fig. 6: Máquina de estados finito del tratamiento del convenio de pila.

La idea a transmitir es que hay registros cuyo valor se ha de preservar durante la ejecución de una subrutina. O bien no se modifican (porque solo se hace lectura de registro o RR) o, si es preciso modificarlos (por escritura en registro o WR), entonces al empezar a ejecutar la subrutina se han de guardar en la zona de memoria de pila (con una escritura en memoria o WM) y antes de terminar se ha de recuperar el valor guardado (con una lectura desde memoria o RM). Es posible guardar un mismo registro en distintas posiciones de memoria (WM!=) siempre que

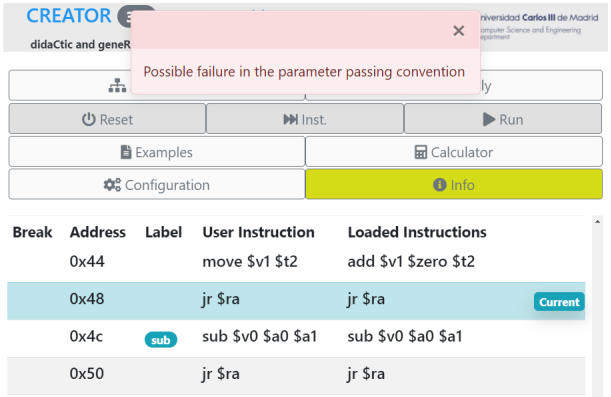

Fig. 7: Mensaje de error por incumplimiento del convenio.

se recupere el valor primeramente guardado.

Con toda esta información sería posible hacer un análisis de todas las posibles secuencias de operaciones indicadas formando un árbol como el mostrado en la Figura 8. A partir de dicho árbol se puede diseñar una máquina de estados finito como la mostrada en la Figura 6.

Como parte de la API que se ofrece en CREATOR que puede usarse en el código que define las instrucciones de la arquitectura, existen unas llamadas que permiten indicar las transiciones de esta máquina de estado finita. De esta manera, sería posible añadir a una arquitectura el soporte que indique a las personas que usen CREATOR que hay algún problema con el uso del convenio de paso de parámetros.

En caso de existir un incumplimiento del convenio que se haya definido en la arquitectura, se mostrará una notificación como la que se puede ver en la Figura 7

IV. Conclusiones y trabajos futuros

Este trabajo describe CREATOR, que es un nuevo simulador altamente intuitivo y portable que permite mejorar la experiencia de aprendizaje y uso de ensamblador MIPS o RISC-V.

CREATOR permite definir y editar desde la propia herramienta las instrucciones del juego de instrucciones a usar (su formato, el efecto de la instrucción, etc.). También permite definir y editar elementos hardware como bancos de registros, directivas, pseudoinstrucciones, etc. Partiendo de las instrucciones de MIPS o RISC-V se pueden ampliar o reducir los juegos de instrucciones de manera que se puedan diseñar prácticas de laboratorio más personalizadas a los objetivos docentes deseados.

Disponer de una versión en línea de mandatos facilita la accesibilidad a estudiantes con dificultades de visión dado que es posible conectar la salida del simulador a un lector de pantalla. Además, también facilita la creación de baterías de pruebas que ayudan a los profesores en la corrección.

Hay varias líneas de trabajos futuros en las que actualmente se está trabajando.

Al respecto de los juegos de instrucciones existentes (MIPS y RISC-V), se está empezando el estudio

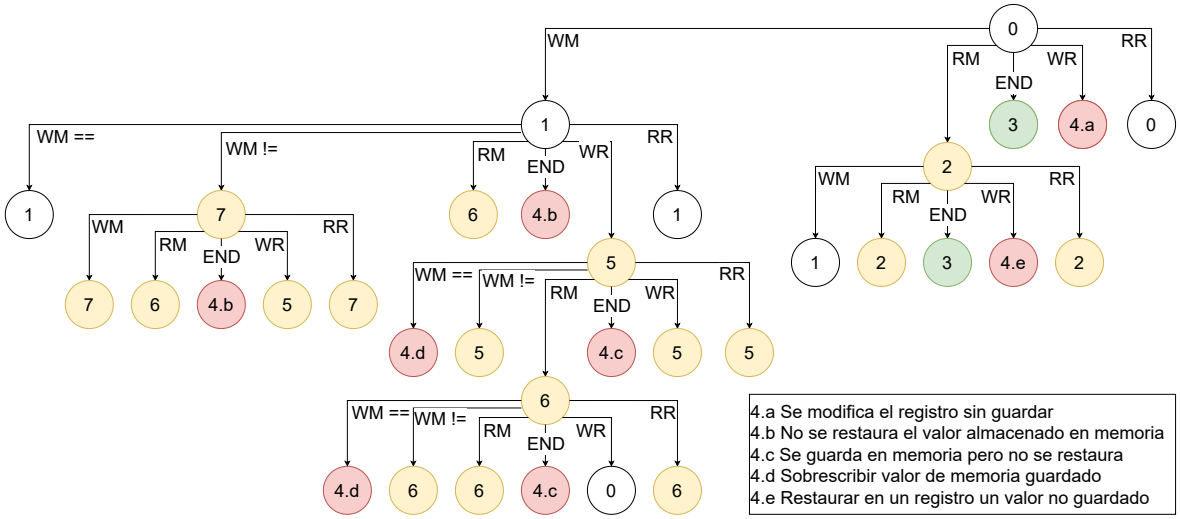

Fig. 8: Árbol de principales posibles pasos en el tratamiento del convenio de pila.

de añadir nuevos conjuntos de instrucciones como, por ejemplo, los de ARM de 32 bits.

En cuanto hardware usado en la simulación, se está estudiando añadir la posibilidad de ejecutar con segmentación (*pipeline*) en las CPU usadas, así como añadir una memoria caché configurable que permita definir los principales aspectos (tipo, tamaños de línea, número de líneas, política de reemplazo, etc.).

Para el entorno de ejecución, se está trabajando en empaquetar el simulador como aplicación nativa para dispositivos móviles Android y iOS (por ejemplo, usando Apache Cordova) para permitir que pueda usarse sin conexión permanente a Internet.

Referencias

[1] Félix García Carballeira, Jesús Carretero Pérez, José Daniel García Sánchez, and David Expósito Singh, *Problemas resueltos de estructura de computadores, segunda edición*, vol. 1, pp. 1–307, Ediciones Paraninfo, 2015.

[2] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Amsterdam, 5 edition, 2012.

[3] David A. Patterson and John L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2017.

[4] Bruno Nova, João C. Ferreira, and António Araújo, "Tool to support computer architecture teaching and learning," in *2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)*, 2013, pp. 1–8.

[5] Belen Bermejo, Carlos Guerrero, Isaac Lera, and Catalina Lladó, *Un simulador de arquitectura MIPS para el estudio del procesamiento paralelo de instrucciones*, p. 233–239, Universitat Oberta La Salle, Jul 2015.

[6] Manuel Rivas Pérez, Manuel Domínguez Morales, Francisco Gómez Rodríguez, Alejandro Linares Barranco, Gabriel Jiménez Moreno, and Antón Civit Balcells, "Diseño e implementación de un simulador software basado en el procesador MIPS32," *Enseñanza y aprendizaje de ingeniería de computadores: Revista de Experiencias Docentes en Ingeniería de Computadores*, , no. 5, pp. 79–104, 2015.

[7] James R. Larus, "SPIM," *spimsimulator.sourceforge.net*. Accessed Jun. 14, 2021. [Online], 2016.

[8] Kenneth Vollmar and Pete Sanderson, "MARS: an education-oriented MIPS assembly language simulator," in *SIGCSE*, 2006, vol. 6, pp. 239–243.

[9] Benjamin Landers, "RARS," *https://github.com/TheThirdOne/rars*. Accessed Jun. 14, 2021. [Online], 2021.

[10] Irina Branovic, Roberto Giorgi, and Enrico Martinelli, "WebMIPS: a new web-based MIPS simulation environment for computer architecture education," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*. 2004, p. 19, ACM.

[11] Inclusive Design Research Centre, "Achecker," *https://achecker.ca/checker/*. Accessed Jun. 14, 2021. [Online], 2020.