Integración del simulador CREATOR con hardware RISC-V: caso de estudio con microcontrolador ESP32

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

Resumen— **Actualmente existen multitud de simuladores de lenguaje ensamblador que permiten a los estudiantes ver y comprender cómo ejecutan los programas ensamblador. Sin embargo, estos simuladores ocultan a los estudiantes todas las implicaciones en términos de rendimiento, consumo de memoria o consumo de energía que conlleva ejecutar estos programas sobre un *hardware* real.**

Por ello, en este trabajo se presenta una nueva funcionalidad desarrollada en el simulador CREATOR que permite cargar y ejecutar el código implementado en el propio simulador en un microcontrolador. Permitiendo a los estudiantes comprender cómo ejecutan los programas ensamblador en un *hardware* real, así como ser una primera toma de contacto en la programación de microcontroladores.

Para este trabajo, se ha utilizado como caso de uso, el lenguaje ensamblador RISC-V, integrándolo y probándolo inicialmente con el microcontrolador Espressif ESP32-C3, lo que permite a los estudiantes ver cómo funcionan sus programas sobre un dispositivo.

Palabras clave— **RISC-V, ESP32-RISCV, simulador, ensamblador, Arquitectura de Computadores.**

I. Introducción

El trabajo presentado describe el diseño y desarrollo de una nueva funcionalidad en el simulador educativo CREATOR [1] [2]. Este simulador es de código abierto y su código fuente se puede encontrar en su repositorio de GitHub [3]. Esta nueva funcionalidad permite que el código ensamblador escrito, compilado y depurado en el simulador pueda ser ejecutado en un dispositivo *hardware* real. Concretamente, para este trabajo, se va a utilizar como caso de uso un microcontrolador ESP32-C3.

Cabe destacar que CREATOR se ha diseñado para ser independiente del *hardware* utilizado, es decir, permite simular el código ensamblador sobre la propia herramienta sin la necesidad de que haya un microcontrolador conectado.

A pesar de la existencia de múltiples simuladores de lenguaje ensamblador, estos no permiten a los estudiantes ver cómo sus programas pueden ser llevados y ejecutados sobre un *hardware* real. Esto provoca que en muchas ocasiones los estudiantes no sean conscientes de las implicaciones en términos de rendimiento, control de gasto de energía y consumo de memoria, entre otros, que su código tiene cuando ejecuta en un *hardware* real.

[1]Departamento de Informática, Universidad Carlos III de Madrid (UC3M), e-mail: {dcamarma,fgcarbal,acaldero,edelpozo}@inf.uc3m.es

[2]https://creatorsim.github.io/

[3]https://github.com/creatorsim/creator

Por otra parte, los entornos de desarrollo que trabajan con *hardware* real como un microcontrolador, habitualmente son entornos profesionales complejos de utilizar o bien son entornos para aprendizaje basados en Arduino con UIFlow [2] o Python. Hasta donde los autores conocen, no existe un entorno intermedio que permita aprender de forma sencilla ensamblador y que, al mismo tiempo, permita ejecutar el código desarrollado sobre *hardware* real, combinando sencillez y profesionalidad a la vez.

Por todo ello, el objetivo de este trabajo es que los estudiantes dispongan de una integración entre el simulador y el *hardware* asequible, fácil y familiar en la que puedan ejecutar sus ejercicios de ensamblador RISC-V y comprender el impacto de su ejecución sobre un dispositivo.

Además, cabe señalar, que el uso de RISC-V [3] permite trabajar con un *hardware* abierto y emergente. Su similitud en ciertos aspectos a MIPS [4] (uno de los ensambladores más usados en docencia) permite una fácil transición a un nuevo procesador diseñado teniendo en cuenta los requisitos actuales y futuros de la arquitectura de computadores.

Asimismo, dado que se ha elegido para el caso de uso presentado en este trabajo como *hardware* un microcontrolador, el precio es más asequible respecto al de las placas SBC (*Single Board Computer*) parecida a la Raspberry Pi, cuyos precios actualmente son muy elevados. Además, al estar basado en un microcontrolador, muchos estudiantes estarán familiarizados con el uso de estos dispositivos y su entorno de trabajo, al haber utilizado anteriormente los microcontroladores basados en Arduino.

El resto del documento se estructura de la siguiente forma: la Sección II describe el Estado del Arte; la Sección III presenta las adaptaciones realizadas en CREATOR para permitir su integración con el *hardware*. En la Sección IV se presenta un caso de uso con un microcontrolador ESP32-C3. Por último, la Sección V presenta las principales Conclusiones y Trabajos Futuros.

II. Estado del arte

Actualmente, como se puede ver en [5], existen diferentes herramientas didácticas basadas en el ensamblador RISC-V. Las más conocidas son Jupiter [6], RARS [7] y Venus [8], pero también existen otras herramientas que están especialmente diseñadas para simular la ejecución de ensamblador con un *pipeline* de 5 etapas como son Ripes [9] y WebRISC-V [10].

Por último, cabe destacar RVfpga [11], que es un curso de RISC-V que hace uso de *hardware*. Los usuarios al finalizar estos cursos, según los autores, tendrán un sistema RISC-V en funcionamiento y experiencia práctica en la exploración y el uso de RISC-V SoC y la cadena de herramientas RISC-V, incluidos compiladores y simuladores.

Jupiter [6], según sus autores, es un simulador multiplataforma (Linux, macOS y Windows) educativo de RISC-V (RV32IMF) implementado en Java. Este simulador permite ejecutar programas ensamblador escritos en varios ficheros, permitiendo una mayor modularidad en el código. Además, este simulador dispone de dos modos de operación: interfaz gráfica y línea de órdenes. Sin embargo, este simulador al estar basado en Java dificulta su uso en los dispositivos y no puede ser ejecutado en dispositivos móviles como *tablets*, que cada día son más usadas ya que son fáciles de transportar.

Otro ejemplo es el simulador de RISC-V llamado RARS [7] que está desarrollado utilizando como base el simulador de MIPS-32 MARS, pero no integrado con él. Por lo que tiene funcionalidades y limitaciones muy similares a las de MARS. Entre estas limitaciones está, como ocurre con el simulador anterior, la dificultad para desplegar el simulador al estar basado en Java en vez de en HTML5, el juego de instrucciones disponible no es editable y su accesibilidad es limitada, entre otras.

Venus [8] es un simulador educativo de RISC-V que dispone de una versión web y de una versión implementada en Java. Este simulador permite simular y depurar los programas ensamblador implementados en el editor del propio simulador, así como visualizar el estado de la memoria principal. Sin embargo, como ocurre con los simuladores mencionados anteriormente solo implementa un subconjunto limitado de la ISA de RISC-V (RV32IM) y no puede ser editado.

Además de los simuladores descritos anteriormente existen otros simuladores que permiten simular el programa ensamblador con un *pipeline*. Entre estos se encuentra la herramienta Ripes [9] que es un simulador basado en un *pipeline* de 5 etapas del ensamblador RISC-V que comenzó a desarrollarse en 2016. Cabe destacar que esta herramienta también permite simular la memoria caché. Sin embargo, este simulador está desarrollado en C++ y su interfaz gráfica se basa en la biblioteca Qt. Lo que imposibilita su uso en dispositivos móviles (*smartphones* y *tablets*) ya que solo puede ser usado en los sistemas operativos Linux, macOS y Windows.

Por último, existe WebRISC-V [10] que es un simulador web que ejecuta programas escritos en RISC-V de 32 bits. Al igual que el simulador Ripes, este simulador ejecuta los programas ensamblador con segmentación (*pipeline*) de 5 etapas. Sin embargo, este simulador implementa un subconjunto muy limitado del juego de instrucciones de RISC-V y tampoco permite editar este subconjunto para añadir nuevas instrucciones.

Tras estudiar los simuladores anteriores CREATOR facilita el aprendizaje porque reúne las características recomendadas:

- Es multiplataforma y no precisa de instalación (solo precisa de un navegador web).
- Dispone del juego de instrucciones RISC-V y MIPS32, pero el juego de instrucciones es personalizable.
- Dispone de interfaz gráfica, así como ejecución en línea de comandos (permitiendo la ejecución de correctores automáticos).
- Muestra errores de compilación, errores de ejecución y errores en el uso del convenio de paso de parámetros.

Todas estas características permiten a los estudiantes aprender lenguaje ensamblador de forma iterativa en un único simulador, permitiendo una fácil transición entre diferentes lenguajes ensamblador.

Por otro lado, en cuanto al *hardware* que se puede utilizar para ejecutar los programas RISC-V desarrollados en CREATOR, encontramos dos posibilidades principales: bien usar un SBC o bien un microcontrolador.

En cuanto a las SBC, es muy conocida la Raspberry Pi como plataforma para aprendizaje usando el procesador ARM. En el caso de RISC-V, la empresa SiFive ofrece una solución muy similar basada en procesador RISC-V llamada HiFive1 Rev B [12]. Dicha placa permite ejecutar un sistema operativo Linux completo, de forma que es posible, por ejemplo, además de ejecutar programas RISC-V, arrancar un servicio web que permita interactuar con el simulador CREATOR. Además, SiFive dispone también de placas como la Vision Five 2 [13] con GPU integrada. Sin embargo, las SBC tienen algunos inconvenientes en cuanto a su uso en una clase de laboratorio en docencia. El coste de una placa puede representar un problema para los estudiantes, la crisis de microchips que se ha acentuado debido a la pandemia de COVID-19, hace que la compra de estos dispositivos sea todavía más cara, tarde bastante en poder servirse o incluso no existir *stock*. Además, estas placas necesitan una fuente de alimentación para poder funcionar, hay que tener una tarjeta MicroSD con el sistema operativo, conexión a red, etc. lo que obliga a que la clase de laboratorio esté acondicionada.

Otra alternativa son los microcontroladores. Los microcontroladores más conocidos son los usados junto con la plataforma Arduino. Estos permiten la programación tanto en UIFlow [2] como en Python, permitiendo transferir el código binario al microcontrolador para su ejecución en el Arduino. Existen diversos fabricantes de microcontroladores basados en RISC-V, destacando Espressif [14] que dispone de muchos ejemplos y documentación en la plataforma GitHub y en su propia página Web. Los microcontroladores respecto a las SBC son más económicos, facilitando su acceso a los estudiantes, los tiempos de espera del envío generalmente es menor, no precisan de alimentación adicional a la del USB y, además, la

gran mayoría de los estudiantes han utilizado alguna vez Arduino, por lo que están más familiarizados con este *hardware*.

III. Integración CREATOR-*hardware*

En esta sección se va a describir cómo se ha llevado a cabo la integración del simulador CREATOR para permitir el cargado y ejecución de los programas ensamblador implementados en el simulador sobre un microcontrolador, así como los pasos que hay que realizar para su puesta en marcha (prerrequisitos).

A. Diseño de la ejecución en hardware real

Para poder llevar a cabo la carga del programa ensamblador desarrollado en CREATOR sobre un dispositivo *hardware* se han tenido que realizar pequeños cambios en el simulador respecto a la versión inicial [1]. Estos cambios afectan especialmente a la interfaz gráfica y a la implementación de un nuevo servicio web. Este servicio web está implementado en Python 3, y será el encargado de comunicarse con los *drivers* de la placa a utilizar. Para ello, se ha utilizado el *framework* Flask de Python 3 que facilita la implementación de servicios web.

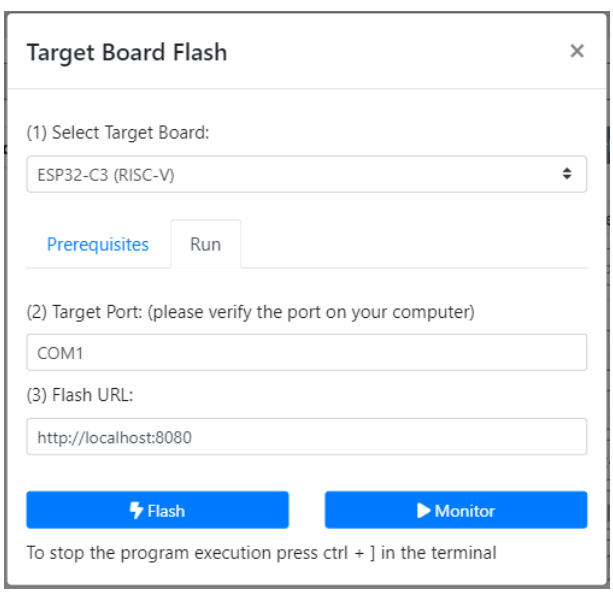

Fig. 1: Formulario de cargado y ejecución de un programa en el microcontrolador.

Respecto a la interfaz de usuario, en CREATOR se dispone de un cuadro de diálogo (ver Figura 1) que permite a los usuarios definir tres parámetros imprescindibles para el correcto cargado del programa en el microcontrolador:

- Modelo del microcontrolador: indica el modelo exacto del *hardware* que se va a utilizar para ejecutar el programa. Por defecto, el modelo será ESP32-C3 de RISC-V.
- Puerto de conexión: define el puerto al que se ha conectado el *hardware*. Como este puerto depende del sistema operativo, por defecto, CREATOR es capaz de detectar el sistema operativo en el que se ejecuta el simulador y poner el puerto que se utiliza por defecto en este.
- URL del servicio web: indica la URL en la que está ejecutando el servicio web, que, por defecto, será: `http://localhost:8080`.

Como se puede ver en la Figura 1 se pueden llevar a cabo dos acciones: *flash* (cargado del programa en el microcontrolador) y *monitor* (visualización de la ejecución del programa cargado en el microcontrolador). Cuando se pulsa alguno de estos botones CREATOR se conectará con el servicio web desarrollado realizando las acciones descritas en la Figura 2.

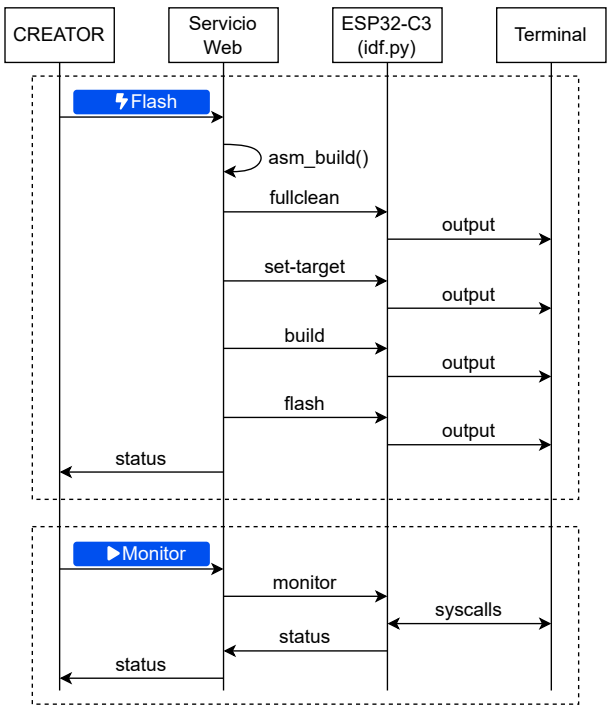

Fig. 2: Intercambio de operaciones entre CREATOR, el servicio web y el *driver* del microcontrolador ESP32-C3.

Por un lado, está el método *flash* que envía al servicio web el modelo del microcontrolador, el puerto al que se ha conectado este y el programa ensamblador que va a ser compilado y cargado en el microcontrolador.

Para ello, en primer lugar, el servicio web guarda el programa ensamblador en un archivo temporal y realiza un preprocesado del código ensamblador para poder emular las llamadas al sistema implementadas en CREATOR en el propio microcontrolador. La Tabla I muestra las llamadas al sistemas en CREATOR, donde la columna *Id.* representa el código de la llamada (que se pasa en el registro a7), la columna *Argumentos* representa el resto de valores a pasar en los registros además del código de llamada, y la columna *Resultado* representa los registros y valores que las llamadas pueden devolver. Este preprocesado es muy importante, ya que permite que el resultado de ejecutar estas llamadas al sistema sea el mismo en el simulador y en el microcontrolador, ya que estas pueden diferir puesto que estos dispositivos no siguen estrictamente los convenios de los lenguajes ensam-

blador y ejecutan en modo protegido.

Tabla I: Llamadas al sistema implementadas en CREATOR.

Llamada al sistema	Id.	Argumentos	Resultado
Print integer	1	a0 = integer	
Print float	2	fa0 = float	
Print double	3	fa0 = double	
Print string	4	a0 = dir. string	
Read integer	5		integer en a0
Read float	6		float en fa0
Read double	7		double en fa0
Read string	8	a0 = dir. string a1 = longitud	
Sbrk	9	a0 = longitud	dir. en a0
Exit	10		
Print char	11	a0 = char ASCII	
Read char	12		char en a0

Tras realizar este preprocesado, el servicio web se encarga de interactuar con los *drivers* específicos del microcontrolador, realizando el proceso de compilación y cargado de datos en el chip de memoria del microcontrolador. A continuación, se devuelve a CREATOR el resultado (*status*) de realizar todos estos pasos para que el simulador pueda devolver un mensaje de error en caso de que algún paso haya fallado.

Por otro lado, el método *monitor* se encarga de visualizar la ejecución del último programa ensamblador cargado en el microcontrolador con el método *flash*. Además, también, permitirá interactuar con el terminal del ordenador que ejecuta el servicio web para poder realizar las diferentes llamadas al sistema, como imprimir valores por pantalla o leer valores desde teclado.

Como ocurre con el método *flash* el resultado global de la ejecución (*status*) se envía de vuelta a CREATOR para mostrar retroalimentación al usuario del resultado de la ejecución.

B. Prerrequisitos de la ejecución en hardware real

Para poder utilizar el soporte de *hardware* disponible en CREATOR es necesario seguir una serie de pasos. Estos pasos se pueden dividir en dos bloques principales: por un lado, la instalación del *software* necesario para trabajar con el microcontrolador y, por otro lado, la ejecución de este *software*.

La instalación del *software* necesario dependerá del Sistema Operativo utilizado en el ordenador donde se va a instalar y ejecutar el *software*, que podrá ser Linux, macOS o Windows, así como del modelo de microcontrolador que queremos utilizar.

Partiendo de la premisa de que está Python 3 instalado en el sistema operativo hay que instalar los paquetes de Python 3 necesarios para ejecutar el servicio web, así como el *software* de desarrollo del microcontrolador para dicho sistema operativo.

Una vez hemos seleccionado el *hardware* a utilizar en CREATOR, los pasos que hay que realizar serán mostrados en el propio simulador, como se puede ver en la Figura 3.

La primera vez que trabajemos con este microcontrolador tendremos que realizar los siguientes pasos:

1. Instalar el *software* de desarrollo del microcontrolador.
2. Instalar los paquetes `flask` y `flask_cors` de Python 3.
3. Descargar desde CREATOR el archivo zip con el *driver* asociado al microcontrolador. Este archivo comprimido contiene un proyecto para el *hardware* seleccionado, así como el servicio web comentado al principio de la sección.
4. Descomprimir dicho archivo zip.

Cada vez que ejecutemos el servicio web, los pasos que hay que realizar son:

1. Cargar las variables de entorno asociadas al microcontrolador. Por ejemplo, en ESP32:
   `. $HOME/esp/esp-idf/export.sh`
2. Cambiar el directorio de trabajo al directorio donde están los archivos descomprimidos del *driver*. Por ejemplo, en ESP32:
   `cd $HOME/creator/esp32c3/`
3. Ejecutar el servicio web. Por ejemplo, en ESP32:
   `python3 gateway.py`

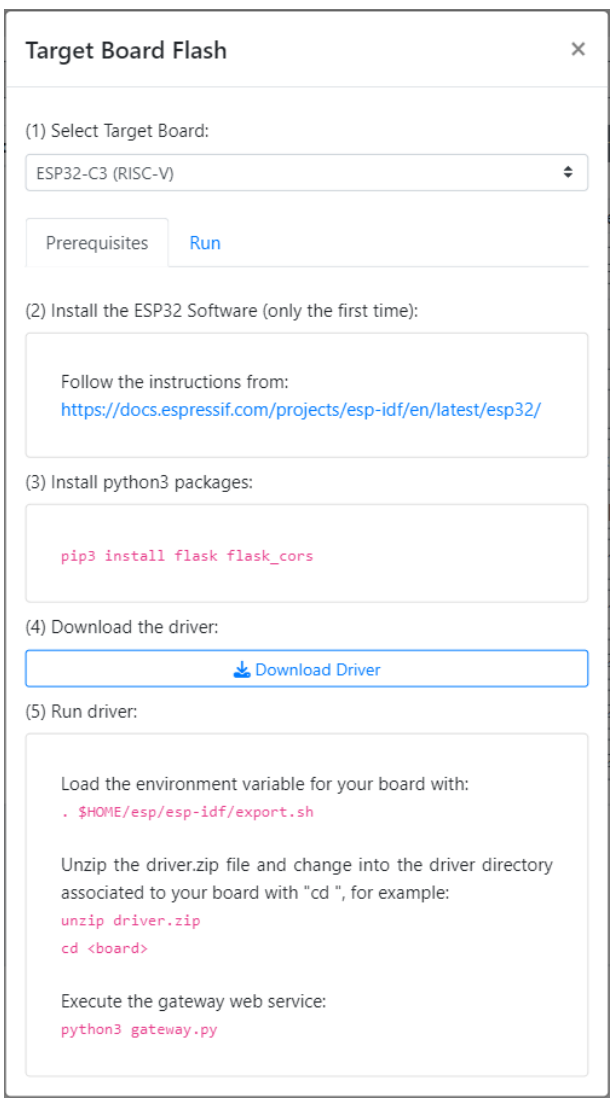

Fig. 3: Prerrequisitos para utilizar el soporte de *hardware* de CREATOR.

```
.text
factorial:
   # crear marco de pila
   addi sp, sp, -12
   sw   ra, 8(sp)
   sw   fp, 4(sp)
   addi fp, sp, 4

   li   x5, 2
   bge  a0, t0, b_else

   # if (a0 < 2):
   #   a0 = 1
   #   return a0
   li   a0, 1
   beq  x0, x0, b_efs

   # else:
   #   a0=a0*factorial(a0-1)
   #   return a0
 b_else:
   sw   a0, -4(fp)
   addi a0, a0, -1
   jal  x1, factorial
   lw   t1, -4(fp)
   mul  a0, a0, t1

 b_efs:
   # finalizar marco de pila
   lw   ra, 8(sp)
   lw   fp, 4(sp)
   addi sp, sp, 12

   # return a0
   jr ra
```

```
main:
    # crear marco de pila
    addi sp, sp, -20
    sw ra, 16(sp)
    sw s0, 12(sp)
    sw s1,  8(sp)
    sw s2,  4(sp)
    sw s3,  0(sp)

    # s=0
    li s1, 0
    li s0, 3
    # while (s1 < 3) {
w1: bge  s1, s0, w1_end

    # s2 = get_cycles()
    rdcycle s2
    # factorial(10)
    li  a0, 10
    jal x1, factorial
    # s3 = get_cycles()
    rdcycle s3

    # print(s3-s2)
    sub a0, s3, s2
    li a7, 1
    ecall

    # s1++
    addi s1, s1, 1
    beq  zero, zero, w1
    # }

w1_end:
    # restaurar pila
    lw s3,  0(sp)
    lw s2,  4(sp)
    lw s1,  8(sp)
    lw s0, 12(sp)
    lw ra, 16(sp)
    addi sp, sp, 20
    jr ra
```

Fig. 4: Programa RISC-V que mide de forma aproximada el número de ciclos para calcular el factorial de 10 con recursividad e imprime dicho número de ciclos.

Una vez se han seguido los pasos anteriores tendremos ejecutando el servicio web y CREATOR se podrá conectar con él para cargar y ejecutar el programa ensamblador deseado.

IV. Caso de uso: Espressif ESP32-C3

Para probar la nueva funcionalidad de CREATOR presentada en este trabajo, se ha usado como caso de uso el microcontrolador ESP32-C3 (como el mostrado en la Figura 5).

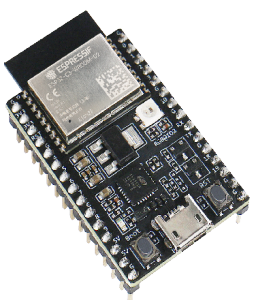

Fig. 5: Microcontrolador ESP32-C3-DevKitC-02 basado en RISC-V [14].

Este microcontrolador tiene una CPU RISC-V de 32 bits con un único core de hasta 160 MHz, 400 KB de SRAM, 384 KB de ROM y una memoria caché asociativa de 16KB con un tamaño de bloque de 32 bytes [15]. Cabe destacar que este microcontrolador no tiene soporte de operaciones en coma flotante.

Para llevar a cabo este caso de uso, se han utilizado el programa RISC-V mostrado en la Figura 4. Este programa mide de forma aproximada el número de ciclos de reloj que se necesitan para calcular el factorial de 10 (la subrutina factorial está implementada de forma recursiva). Una vez realizada la medición se imprime el número de ciclos de reloj. El cálculo de ciclos de ejecución se lleva a cabo 3 veces, esto permite ver a los estudiantes como la memoria caché del microcontrolador permite reducir el número de ciclos necesarios para la ejecución de un mismo código al no tener que realizar accesos a memoria principal la segunda y tercera vez que se ejecuta.

En primer lugar, una vez se ha implementado el programa ensamblador usando el editor de código de CREATOR, hay que compilarlo para verificar que no existen errores en el código. Si la compilación no devuelve ningún error, el siguiente paso será ejecu-

tar este programa en el simulador para verificar qué funciona correctamente antes de llevarlo al dispositivo *hardware*.

Una vez hemos verificado que la ejecución es correcta utilizando CREATOR, procederemos a cargar el programa en el microcontrolador haciendo uso del formulario mostrado en la sección anterior (ver Figura 1). Concretamente, ejecutaremos la acción *flash*.

Cuando se ejecuta la acción de *flash* en el servicio web, como se explicó en la sección anterior, se realiza un preprocesado sobre el código ensamblador implementado para que las llamadas al sistema funcionen de igual forma que en CREATOR y el resultado sea el mismo. Una vez se ha adaptado el código ensamblador, el servicio web carga el programa ensamblador en el microcontrolador mostrándose por pantalla el proceso de cargado, como se puede ver en la Figura 6.

```
Flash will be erased from 0x00000000 to 0x00005fff...
Flash will be erased from 0x00010000 to 0x00048fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 20656 bytes to 12717...
Writing at 0x00000000... (100 %)
Wrote 20656 bytes (12717 compressed) at 0x00000000 in 0.8 seconds
(effective 204.1 kbit/s)...
Hash of data verified.
Compressed 231584 bytes to 114382...
Writing at 0x00010000... (14 %)
Writing at 0x0001eefe... (28 %)
Writing at 0x0002587c... (42 %)
Writing at 0x0002d138... (57 %)
Writing at 0x000345bf... (71 %)
Writing at 0x0003a701... (85 %)
Writing at 0x0004156f... (100 %)
Wrote 231584 bytes (114382 compressed) at 0x00010000 in 3.6 second
s (effective 516.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (ef
fective 279.6 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
```

Fig. 6: Proceso de cargado del programa en el microcontrolador.

Por último, una vez se ha cargado correctamente el programa ensamblador en el microcontrolador, podremos visualizar su ejecución en el dispositivo. Para ello, se utilizará la acción *monitor* del cuadro de diálogo mostrado en la Figura 1. Durante la ejecución de esta acción, en la terminal donde se ejecuta el servicio web se irá mostrando la salida del programa. Y, también, será posible introducir valores por teclado si el programa realiza alguna llamada al sistema de lectura de teclado.

En el Listado 1 se puede ver el resultado de la ejecución del programa ensamblador presentado en la Figura 4. Se puede destacar que la primera vez que se calcula el factorial de 10 se necesitan 1382 ciclos de reloj, mientras que en los cálculos posteriores se necesitan solamente 220 ciclos. Esto se debe a que el código a ejecutar se encuentra en memoria caché y no es necesario leerlo de memoria principal, reduciendo el número de ciclos que necesita la subrutina `factorial` para ejecutarse.

Listado 1: Ciclos de las 3 ejecuciones del factorial de 10.

```
Started program...
-------------------
>1382
>220
>220
Finished program: 53638 cycles
-------------------
```

V. Conclusiones

En este artículo se ha presentado el diseño y desarrollo realizado en el simulador CREATOR para permitir a los estudiantes ejecutar sus programas ensamblador en un microcontrolador real desde CREATOR, así como un caso de uso utilizando el microcontrolador ESP32-C3. Cabe destacar, que el diseño modular de la interfaz de CREATOR ha permitido añadir esta nueva funcionalidad de forma rápida y segura.

Como principales conclusiones de los primeros resultados, podemos destacar que esta nueva funcionalidad añadida en CREATOR permite a los estudiantes probar su código ensamblador usando *hardware* real, ayudándoles a ser conscientes del impacto de la ejecución de su código en términos de rendimiento y gasto energético, como se ha podido ver en el caso de uso. La elección de un microcontrolador para desarrollar esta integración permite no excluir a personas por factores económicos al ser más accesibles que las SBC. Además, su semejanza con Arduino favorece el acercamiento a un entorno de trabajo conocido por estudiantes que hayan trabajado previamente con este microcontrolador.

Como principales líneas de trabajos futuros se plantea la integración con más modelos de microcontroladores compatibles con RISC-V así como MIPS-32. También se quiere simplificar la puesta en marcha del entorno en diferentes plataformas, usando una imagen de contenedor *docker* por ejemplo. Otra línea de trabajo futuro consiste en simular en CREATOR los diferentes GPIO del microcontrolador de forma que se puedan simular operaciones de E/S antes de ejecutarlas directamente sobre el microcontrolador. También, se pretende realizar casos de uso en los que se analice el consumo de energía, así como de memoria.

Agradecimientos

Este trabajo ha sido parcialmente apoyado por la 20ª Convocatoria de Apoyo a Experiencias de Innovación Docente del curso 2022-2023 de la Universidad Carlos III de Madrid.

Referencias

[1] *CREATOR: Simulador didáctico y genérico para la programación en ensamblador*. Zenodo, July 2021.

[2] Puput Dani Prasetyo Adi and Akio Kitagawa, "A review of the blockly programming on m5stack board and mqtt based for programming education," in *2019 IEEE 11th International Conference on Engineering Education (ICEED)*, Nov 2019, pp. 102–107.

[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] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Amsterdam, 5 edition, 2012.
[5] RISC-V.org, "RISC-V.org Exchange," https://riscv.org/exchange/?_sft_exchange_category=software. Accedido el 19-05-2023. [Online], 2023.
[6] Andrés Castellanos, "Jupiter," https://github.com/andrescv/Jupiter. Accedido el 19-05-2023. [Online], 2023.
[7] Benjamin Landers, "Rars," https://github.com/TheThirdOne/rars. Accedido el 14-05-2023. [Online], 2021.
[8] Keyhan Vakil Stephan Kaminsky, "Venus," https://github.com/ThaumicMekanism/venus. Accedido el 19-05-2023. [Online], 2023.
[9] Morten B Petersen, "Ripes: A visual computer architecture simulator," in *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*. IEEE, 2021, pp. 1–8.
[10] Roberto Giorgi and Gianfranco Mariotti, "WebRISC-V: a web-based education-oriented risc-v pipeline simulation environment," in *ACM Workshop on Computer Architecture Education (WCAE-19)*, Phoenix, AX, (USA), jun 2019, pp. 1–6.
[11] Sarah L. Harris, Daniel Chaver, Luis Piñuel, J.I. Gomez-Perez, M. Hamza Liaqat, Zubair L. Kakakhel, Olof Kindgren, and Robert Owen, "Rvfpga: Using a risc-v core targeted to an fpga in computer architecture education," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 145–150.
[12] SiFive, "SiFive HiFive1 rev-b," https://www.sifive.com/boards/hifive1-rev-b. Accedido el 19-05-2023. [Online], 2023.
[13] starfivetech, "StarFive VisionFive 2," https://www.starfivetech.com/en/site/boards. Accedido el 19-05-2023. [Online], 2023.
[14] Esspressif, "Esspressif ESP32," https://www.espressif.com. Accedido el 19-05-2023. [Online], 2023.
[15] Espressif, "Espressif ESP32-C3 Datasheet," https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf. Accedido el 19-05-2023. [Online], 2023.