Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

Artículo original

Optimizando microcontroladores RISC-V con el uso de fusión de macrooperaciones

Optimizing RISC-V microcontroller with the use of macro operations fusion

Edgardo Alberto Gho[1], Martin Ferreyra Biron[2]

[1] Universidad Nacional de La Matanza, Departamento de Ingeniería e Investigaciones Tecnológicas,
Grupo de investigación en lógica programable, Buenos Aires, Argentina.
[1] Universidad Abierta Interamericana. Facultad de Tecnología informática.
Centro de Altos Estudios en Tecnología Informática. Buenos Aires, Argentina.
egho@unlam.edu.ar
ORCID Id: 0009-0008-0665-4183

[2] Universidad Nacional de La Matanza, Departamento de Ingeniería e Investigaciones Tecnológicas,
Grupo de investigación en lógica programable, Buenos Aires, Argentina.
mferreyra@unlam.edu.ar
ORCID Id: 0009-0006-0960-7900

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

Resumen:

En los últimos años la arquitectura abierta RISC-V ha comenzado a destacar tanto en ámbitos comerciales como académicos, conduciendo esto a modificar dicha arquitectura para obtener mejor eficiencia en la ejecución de programas. RISC-V propone al menos dos formas para lograr un mejor desempeño: utilizar un esquema de segmentación de cauce (pipelining) o realizar fusión de instrucciones, cuya implementación queda a criterio del desarrollador del microcontrolador. En el presente trabajo se realiza un detallado análisis sobre cómo es posible implementar la fusión de instrucciones y fusionar como máximo cuatro de éstas, valiéndose del estándar C ofrecido por RISC-V (RVC), marcando las ventajas y desventajas de llevar esto a cabo, utilizando dos esquemas distintos de implementación de memoria.

Abstract:

In recent years, the open-source architecture RISC-V began to gain prominence in both commercial and academic circles, leading to modification to the architecture to achieve greater efficiency in program execution. RISC-V proposes at least two ways to obtain better performance: Use a pipeline scheme or implement instruction coalescing, which is at the discretion of the microcontroller designer. This document provides a detailed analysis of how it is possible to implement the instruction coalescing and coalesce a maximum of four instructions using the C standard provided by RISC-V (RVC), studying the advantages and disadvantages of doing this using two different memory schemes.

Palabras Clave: *RISC-V, RVC, fusión de macrooperaciones, optimización*

Key Words: *RISC-V, RVC, macro-operations fusion, optimization*

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

I. CONTEXTO

RISC-V fue concebida como una arquitectura abierta, de fines educativos y de investigación, pero rápidamente está siendo adoptada por fabricantes de microcontroladores en productos comerciales y se prevé que su adopción sea aún mayor en el futuro [1]. Esto posiblemente sea atribuible al hecho de ser una arquitectura libre y abierta y, por lo tanto, no es necesario el pago de cánones [2]; dando como resultado que grandes compañías e instituciones académicas [3] utilicen o estén interesadas en esta arquitectura [4]. Las características que ofrece RISC-V posibilitan expandir sus capacidades implementando hardware a medida, haciendo que sea muy atractiva para cualquiera que desee utilizar esta arquitectura. A partir de lo mencionado, se puede concluir rápidamente que la optimización en esta arquitectura depende de aquel que la implemente y, siendo que RISC-V gana cada vez mayor interés en la industria, las optimizaciones comienzan a tener mayor relevancia.

II. INTRODUCCIÓN

El tamaño de las palabras en RISC-V varía dependiendo de la versión implementada, tal es así que si la versión de RISC-V es RV32 los registros y el espacio de direcciones será de 32 bits, pero si la versión es RV64, estos cambian a 64 bits. Lo mencionado no implica que el tamaño de las instrucciones esté circunscrito solamente a esas dos longitudes, ya que, para evitar limitaciones, RISC-V permite extender el tamaño de las instrucciones de a 16 bits admitiendo también 16, 32, 48, 64 bits, entre otras [5]. Sin embargo, al usar RISC-V y desarrollar software para esta arquitectura, se demuestra de manera empírica que entre el 50 y el 60% [5] de las instrucciones podrían ser codificadas solamente con 16 bits, lo que resultaría en un ahorro considerable de entre el 20 y el 30% del tamaño del código, comprimiendo las instrucciones. A partir de esto se crea, para RISC-V, el estándar C (RVC) que permite codificar un selecto grupo de instrucciones en 16 bits, pudiendo este estándar ser utilizado con instrucciones de otro tamaño. Esta forma de comprimir instrucciones no es exclusiva de RISC-V, otras arquitecturas como x86, codifican las instrucciones con tamaños variables, permitiendo representar algunas con una menor cantidad de bits. Por otra parte, arquitecturas como ARM utilizan un modo que permite representar instrucciones en 16 bits (modo Thumb) [6]. Dichas instrucciones son versiones comprimidas de sus representaciones en 32 bits, las cuales el decodificador identifica y convierte, utilizando lógica combinacional, a su equivalente en 32 bits. Esto cumple con el objetivo de reducir el tamaño del código, pero como contrapartida el requisito es forzar al procesador a utilizar el modo Thumb. A tal fin existe un bit de control (bit `T´) que le indica al procesador en que modo se encuentra trabajando. Dicho bit puede cambiar de estado en cualquier momento, pero como contrapartida esto lleva a algo de sobrecarga: para lograr una eficiencia por lo menos igual que al ejecutar una instrucción de 32 bits, se requiere que 2 instrucciones consecutivas puedan ser traducidas de 32 bits a 16 bits [7].

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

En el caso de RISC-V, siendo ésta una arquitectura más moderna que las anteriores, se valió de las experiencias previas: las instrucciones comprimidas en RISC-V se traducen a instrucciones originales de forma transparente y combinacional, sin agregar complejidad al decodificador de instrucciones y sin necesidad de un bit especial para forzar la arquitectura a un modo particular. Esto se resuelve en la microarquitectura, detectando si la instrucción a ejecutar se encuentra comprimida o no en tiempo de ejecución.

Estas características de la arquitectura RISC-V junto con el estándar RVC, hacen a la arquitectura óptima para realizar un tipo de optimización en el rendimiento denominada fusión de operaciones o macrooperación [8], la cual explicaremos a continuación.

III. FUSIÓN DE OPERACIONES

Los sets de instrucciones pueden ser divididos en dos grandes grupos CISC y RISC, el primero hace referencia a computadoras cuyos sets de instrucciones están compuestos por instrucciones complejas, es decir instrucciones que engloban varias operaciones. En cambio, RISC hace referencia a computadoras cuyos sets de instrucciones son reducidos en comparación con CISC y están compuestos por instrucciones sencillas. Ambas filosofías tienen sus ventajas y desventajas y sus diferencias han sido objeto de numerosos análisis durante años, pero si nos enfocamos en una arquitectura como RISC-V, cuyo set de instrucciones es RISC, es común encontrar que en distintos programas ciertas secuencias de instrucciones se repitan frecuentemente. Estas secuencias podrían hacer reconsiderar el hecho de agregar una nueva instrucción que las englobe, incrementando el rendimiento y si bien es posible agregar nuevas instrucciones de diversas formas [9], eso requeriría crear un set de instrucciones con un standard RISC-V propio, que ya no sería compatible con todas las herramientas de desarrollo de RISC-V o ir en desmedro de la eficiencia. La mejor opción sería continuar respetando un standard RISC-V mientras se incrementa el rendimiento. No menos cierto es que en otras arquitecturas como ARM, es posible que se añada una nueva instrucción al detectar estas secuencias, dándole por nombre macrooperación [10] o dicho de otra forma fusión de operaciones. Sin embargo, en RISC-V, es responsabilidad del diseñador del procesador implementar las optimizaciones en la microarquitectura, tratando de detectar qué conjuntos de instrucciones corresponderían a una macrooperación y logrando que en conjunto se ejecuten en menor tiempo que al ejecutarse por separado. Para ejemplificar el concepto de la fusión de operaciones, tomemos una típica instrucción de lenguaje C, en donde se accede al N-ésimo elemento de un vector, guardándose en una variable:

int dato = vector [indice];

Teniendo en cuenta que cada elemento del vector es un número entero de 4 bytes (tamaño de la palabra) y qué índice es un número natural que indica qué palabra debe ser obtenida, se deduce que la dirección desde la que se debe leer el dato se calcula de la siguiente forma:

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

*dirección base+índice*tamaño de la palabra*

Donde *dirección base* es la dirección inicial del vector y *tamaño de la palabra* está representado en bytes. Si deseamos solucionar este cálculo, podemos comenzar tomando el valor de *índice* y multiplicarlo por *tamaño de la palabra,* que en este caso es 4 (debido a la suposición previa de que un entero se representa con 4 bytes), este número significa las posiciones de memoria que se deben saltear desde la dirección base para comenzar a leer el dato, por lo tanto, lo restante es sumar este resultado a la dirección base del vector. Podemos representar esta secuencia con registros llamando:

- RD al registro destino en el cual se deposita el valor leído de memoria
- RS1 como el registro que contiene el índice
- Y RS2 como el registro que contiene la dirección base del vector

Como hemos mencionado, debido a como se representan los números enteros, el índice debe ajustarse multiplicándose por cuatro, o lo que es lo mismo, agregando dos ceros en las posiciones menos significativas del registro RS1. Por lo tanto, la secuencia de operaciones sería:

- Desplazar a la izquierda RS1 en 2 posiciones
- Sumar RS1 (desplazado) y RS2
- Utilizar la suma anterior como dirección de memoria y guardar el contenido en RD.

Como mencionamos, estas operaciones pueden ser englobadas en una instrucción y, de hecho, lo son en ARM con la instrucción LDR RD , [RS2, RS1] [11]

IV. ALINEAMIENTO DE INSTRUCCIONES

Para poder llevar a cabo la fusión de operaciones es necesario comprender que el principal inconveniente radica en que el esquema de ejecución tradicional en orden solo permite ejecutar una sola instrucción a la vez; y para fusionar una secuencia de 3 instrucciones, se requiere que la CPU pueda “observar” las dos instrucciones siguientes. Para lograr esto se debe crear un esquema complejo en la unidad de gestión de memoria (MMU), la cual debería ser diseñada a medida para la fusión de operaciones: la MMU requeriría tener acceso a 3 palabras de 32 bits de memoria, teniendo cada instrucción una longitud de 4 bytes (en el caso de RV32i). Se puede observar fácilmente que la cantidad de palabras que se deben leer de la memoria, para poder detectar si es posible fusionar un conjunto de operaciones, es considerable y esto iría en desmedro de la optimización que podría ofrecer fusionar instrucciones; sin embargo, podemos sortear, en parte, este inconveniente utilizando el estándar RVC de RISC-V: es altamente probable que 2 instrucciones de 16 bits se encuentren en la misma palabra de memoria, o que 4 instrucciones se encuentren juntas en dos palabras memorias contiguas consiguiendo así que la cantidad de palabras que se necesiten leer de memoria disminuya. Sin embargo, los beneficios de utilizar RVC están íntimamente ligados a que el orden de las instrucciones quede alineado a 16 bits, permitiendo el

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

High
⋮
0x1579
0x0517

Low
⋮
0x1000
0x05a9

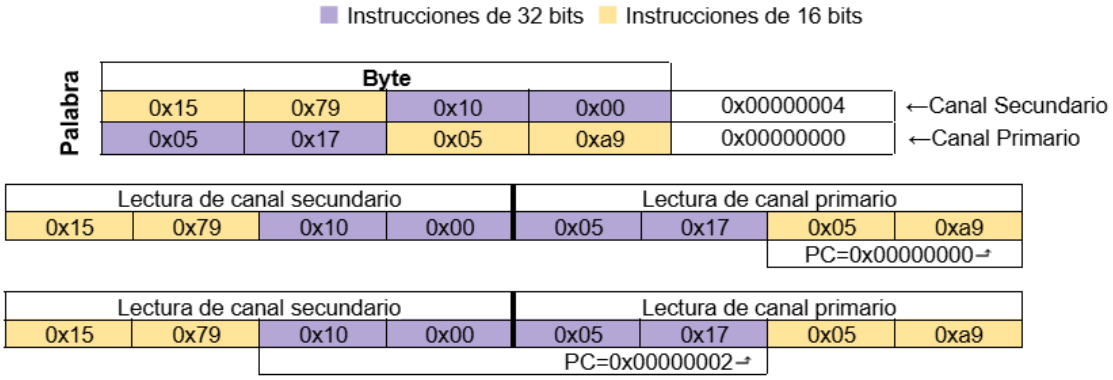
Dirección
⋮
0x00000004
0x00000000

"Fig. 1" Primer esquema de memoria

estándar convivir en la misma memoria tanto instrucciones de 16 bits como de 32 bits. Hacer caso omiso a este inconveniente puede producir fácilmente accesos no alineados que van a requerir un manejo especial. Para solventar este problema existen dos formas de abordarlo: el primero es utilizar un mecanismo donde se utilice un búfer intermedio, pero esto conlleva a pagar penalidades debido a los tiempos de accesos a memoria y, por otro lado, se puede diseñar un hardware de una forma inteligente en la cual sea posible acceder a direcciones de memoria consecutivas sin sufrir penalidades, existiendo también dos esquemas para enfrentar este inconveniente, los cuales analizaremos a continuación. El primer esquema consiste en utilizar dos memorias idénticas de un único canal de direcciones (*single-port)* cuyo tamaño de palabra es de 16 bits. A partir de estas dos memorias conseguiremos realizar un direccionamiento, un poco más complejo, combinando inteligentemente las lecturas de estas dos memorias, según la instrucción esté alineada o no a 32 bits.

Para ejemplificar cómo funciona este esquema, partamos de un programa que se compone de tres instrucciones de assembly de RISC-V, las cuales son 0x05a9;0x10000517 y 0x1579. Si estas instrucciones las colocáramos en una memoria con este esquema, quedarían distribuidas como se muestra en la Fig. 1. Podemos observar que al comenzar el programa y hacer referencia a la dirección de memoria 0x00000000 accedemos a 0x051705a9. La parte alta de esta palabra (0x0517) se encuentra en las direcciones 2 y 3 en la memoria High, mientras que la parte baja (0x05a9) se encuentra en las direcciones 0 y 1 de la memoria Low. Al leer el esquema de codificación de estos 32 bits, se determina que la parte baja posee una instrucción comprimida (0x05a9) y luego de resolver esta instrucción, el PC avanza en 2 bytes. Al no encontrar una instrucción comprimida, la decodificación detecta esta situación y accede a las direcciones 2 y 3 en la memoria High, junto con las direcciones 4 y 5 de la memoria Low. De esta manera se forma una nueva instrucción: 0x10000517 que es la siguiente instrucción a ejecutar, pero debemos notar que la parte baja (0x0517) se encuentra almacenada en la memoria High y que la lógica de direccionamiento no es fácil de implementar. Si bien este esquema permite el acceso no alineado sin penalidad de tiempo, encierra la complejidad adicional de separar la información en dos partes, para así poder inicializar ambas memorias por separado. Por otra parte, el segundo esquema consiste

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)

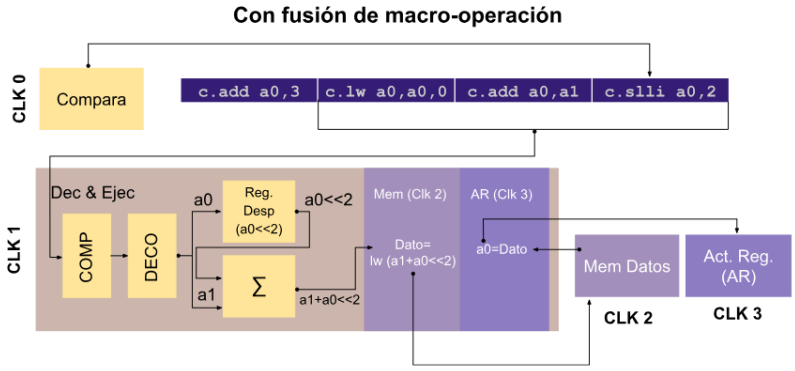


"Fig. 2" Segundo esquema de memoria

en utilizar una única memoria de dos canales de direcciones (*dual port*). Este tipo de memorias brinda dos canales de acceso en paralelo a la información de una única memoria, por lo tanto, brinda dos buses de direcciones que pueden acceder aleatoriamente al contenido de dos direcciones. Con este esquema de memoria no se utilizará una memoria de dos canales de direcciones de palabras de 16 bits, sino una memoria de dos canales de direcciones de palabras de 32 bits, haciendo uso del puerto de dirección principal para leer la dirección actual de la memoria de instrucción, y el puerto de dirección secundario para leer el contenido de la dirección siguiente. Leyendo de esta manera, podemos acceder a 64 bits consecutivos. Para ilustrar mejor este segundo esquema, continuaremos utilizando como ejemplo el programa 0x05a9;0x10000517;0x1579.

Si estas instrucciones las colocáramos en una memoria con este esquema, quedarían distribuidas como se muestra en la Fig. 2. Siendo que PC se encuentra en la dirección 0x00000000, en el canal primario la dirección a la que se accederá es 0x00000000, y en el canal secundario se accederá a 0x00000004. Del canal primario se leerá el contenido 0x051705a9, el cual está comprendido por una instrucción comprimida (0x05a9) en la parte baja y por 0x0517 en la parte alta, que corresponde a la parte baja de una instrucción de 32 bits. Al mismo tiempo, del canal secundario se obtiene 0x15791000, cuya palabra está compuesta por una instrucción comprimida en la parte alta (0x1579) y parte de una instrucción de 32 bits en la parte baja (0x1000). Hasta este momento solo se ha leído la memoria. El PC se encuentra apuntando a 0x00000000 y en dicha posición existe una instrucción de 16 bits, la cual el procesador ejecuta. Al finalizar la ejecución, el PC se incrementa en 2, quedando el mismo en 0x00000002. En esta posición de memoria se tiene una instrucción de 32 bits, aunque solo la parte baja (0x0517), sin embargo, al haber ya leído la palabra 0x00000004 que contiene 0x15791000 se tiene acceso la parte alta de esta instrucción y por lo tanto es posible confeccionar la instrucción de 32, combinando estas lecturas, obteniendo el dato 0x10000517 y evitando la penalización por el acceso no alineado. Se puede observar que de esta forma podemos soportar accesos

Revista Digital del Departamento de
Ingeniería e Investigaciones Tecnológicas
de la Universidad Nacional de La Matanza
ISSN: 2525-1333
Vol.:10-Nro.1 (julio-2025)



"Fig. 3" Implementación de macroinstrucción en RISC-V

no alineados sin tener que separar los datos en dos memorias distintas. Por desgracia, este tipo de memorias son, desde un punto de vista económico, más onerosas que las memorias de un único canal de direcciones, pero este esquema conlleva una ventaja respecto al esquema de memoria de un único canal de direcciones: en este caso es posible saber que sucede en las tres instrucciones siguientes y por lo tanto, fusionar como máximo cuatro operaciones de instrucciones comprimidas.

V. IMPLEMENTACIÓN

Al solucionar el problema de los accesos no alineados, es posible implementar la detección y fusión de instrucciones. Para ejemplificar esto observemos como sería la fusión de operaciones en RISC-V utilizando el ejemplo mostrado en la sección III, en el cual se accedía a
un elemento de un vector de enteros. Esta operación en RISC-V consume 3 instrucciones comprimidas, y suponiendo que en el registro a0 se encuentra el índice y en a1 la dirección base del vector el código en assembly de RISC-V sería

c.slli a0,2
c.add a0,a1
c.lw a0,a0,0

Al implementar la fusión de instrucciones, en RISC-V tendremos que considerar qué acciones tomar en cada una de las cuatro etapas que conforman el ciclo de ejecución de instrucción (Búsqueda, Decodificación & Ejecución, Acceso a memoria y por último, Actualización de registros)

- En la etapa de Búsqueda, sencillamente se obtiene la instrucción c.ssli a0,2 (Clk 1)
- En la etapa de Decodificación & Ejecución, se analizan las siguientes dos operaciones y se detecta si las mismas corresponden a la macroinstrucción que nos propusimos encontrar. Si en esta etapa no se detecta el patrón esperado, la ejecución continúa de forma tradicional, en caso contrario, se realiza la decodificación y utilizando hardware dedicado (no haciendo uso de la ALU) se multiplica el registro a0 por cuatro y al resultado se le suma el contenido del registro a1. Es en esta etapa donde se deben

evitar los accesos no alineados, utilizando alguno de los métodos mencionados en la sección anterior, siendo el más eficiente el método dos (Clk 2)

- La dirección generada en el punto anterior es utilizada para leer una palabra de memoria, siendo esta la etapa de Acceso a memoria (Clk 3)
- Por último, se almacena el dato leído en el registro a0, siendo que esto corresponde a la etapa de Actualización de registros) (Clk 4)

Toda esta implementación se ve reflejada en el diagrama que muestra la Fig. 3. En el mismo se encuentra marcada cada parte del ciclo de instrucción de RISC-V, utilizando los distintos ciclos de reloj que conforman un ciclo de instrucción en esta arquitectura.

VI. CONCLUSIONES

Como se pudo observar en el presente documento, es posible realizar optimizaciones en la arquitectura RISC-V implementando fusión de operaciones. Este tipo de optimizaciones tiene el beneficio de ser transparente sin modificar el estándar empleado, sin embargo, para poder llevar a cabo este tipo de optimizaciones se necesita hacer uso de hardware dedicado y la cantidad de instrucciones que se puedan fusionar dependerá de las instrucciones consecutivas que se puedan analizar. Para poder hacer este análisis es necesario realizar accesos no alineados sin penalizaciones, siendo el uso de memorias de dos canales de direcciones más provechoso frente a técnicas convencionales, permitiendo fusionar hasta 4 instrucciones utilizando el uso del estándar RVC. Es importante aclarar que este tipo de optimizaciones, no afectan globalmente el desempeño, sino que dependerá de los programas que se ejecuten y de si estos utilizan estas instrucciones. En el caso particular de microcontroladores basados en RISC-V, estos pueden verse beneficiados con el uso de la fusión de macrooperaciones aun en el caso de que el esquema de segmentación de cauce (pipelining) no se encuentre implementado.

VII. REFERENCIAS BIBLIOGRÁFICAS

[1] Omdia, “RISC-V adoption will be accelerated by AI, according to new Omdia research”. Consultado: el 3 de mayo de 2025. [En línea]. Disponible en: https://www.prnewswire.com/news-releases/risc-v-adoption-will-be-accelerated-by-ai-according-to-new-omdia-research-302147909.html

[2] Electronics for u, “RISC-V Market Trends And Predictions Till 2030”. Consultado: el 3 de mayo de 2025. [En línea]. Disponible en: https://www.electronicsforu.com/technology-trends/risc-v-market-trends-predictions-till-2030

[3] Mordor Intelligence,“RISC-V Tech Companies - Top Comp any List”. Consultado: el 3 de mayo de 2025. [En línea]. Disponible en: https://www.mordorintelligence.com/industry-reports/risc-v-tech-market/companies

[4] EEJournal, “Why Universities Want RISC-V”, EEJournal. Consultado: el 3 de mayo de 2025. [En línea]. Disponible en:

https://www.eejournal.com/article/why-universities-want-risc-v/

[5] "The RISC-V Instruction Set Manual: Volume I: Unprivileged Architecture". Consultado: el 3 de mayo de 2025. [En línea]. Disponible en: https://courses.grainger.illinois.edu/ece391/sp2025/docs/unpriv-isa-20240411.pdf

[6] S. R. Caprile, *Desarrollo con microcontroladores ARM*, Buenos Aires, Punto Libro, 2012.

[7] "The Thumb Extension". Consultado: el 3 de mayo de 2025. [En línea]. Disponible en: https://www.cs.umd.edu/~meesh/cmsc411/website/proj01/arm/thumb.html

[8] C. Celio, P. Dabbelt, D. A. Patterson, y K. Asanović, "The Renewed Case for the Reduced Instruction Set Computer: Avoiding ISA Bloat with Macro-Op Fusion for RISC-V", 2016, *arXiv*. doi: 10.48550/ARXIV.1607.02318.

[9] E. Gho, "Penalidades en lecturas no alineadas dentro de Microcontroladores RISC-V", *ReDDI*, vol. 7, núm. 1, pp. 1–9, 2022, doi: 10.54789/reddi.7.1.2.

[10] "ARM Cortex-A78 MOPs, UOPs, and Instruction Fetch Pipeline – System on Chips". Consultado: el 10 de mayo de 2025. [En línea]. Disponible en: https://www.systemonchips.com/arm-cortex-a78-mops-uops-and-instruction-fetch-pipeline/

[11] "LDR instructions in Arm Cortex-M - SoC". Consultado: el 10 de mayo de 2025. [En línea]. Disponible en: https://s-o-c.org/ldr-instructions-in-arm-cortex-m/

Recibido: 2025-05-22
Aprobado: 2025-07-04
Hipervínculo Permanente: https://doi.org/10.54789/reddi.10.1.2
Datos de edición: Vol. 10 - Nro. 1 - Art. 2
Fecha de edición: 2025-07-31