**Núcleo RISC-V (Primera versión)**

Aramallo Guantay, Elian Leandro  
Ingeniería Electrónica  
e-mail: aramallog.elian@gmail.com

**RESUMEN:** Se han buscado los conceptos de arquitectura y microarquitectura de computadora para luego profundizar en la arquitectura de RISC-V y su conjunto de instrucciones RV32I. Se hizo hincapié en la descripción de las instrucciones tipo R, I, S, B, U y J para implementarla en la codificación de una ALU.

Teniendo en cuenta algunos bloques diseñados en los proyectos 8 y 9, como ser el registro 32x32 y la Unidad Aritmético Lógica (ALU), se ha propuesto diseñar un datapath multiciclo que permita implementar el conjunto de instrucciones RV32I. Como resultados se obtuvo el diseño de un procesador capaz de recibir instrucciones escritas en un archivo nombrado origen.mem.

**1 INTRODUCCIÓN**

Para el desarrollo de este proyecto se requiere saber conceptualmente a qué se llama arquitectura de computadora y qué es la microarquitectura de computadora. A continuación se desarrollan dichos conceptos y, además, se presenta la arquitectura RISC-V para luego implementar sus instrucciones en el procesador que se va a desarrollar. De este último, al final, sólo se ejecutará la simulación; es decir, no se hará la implementación en un hardware, lo cual correspondería a la microarquitectura (que se definirá más adelante).

**2 DESARROLLO DE CONCEPTOS INVESTIGADOS**

**2.1 ARQUITECTURA DE COMPUTADORA**

La arquitectura de computadora es la perspectiva que tiene el programador de una computadora, en otras palabras, es el nivel más alto de diseño de una computadora desde el punto de vista lógico, principalmente está orientado para el programador. Está definida por el conjunto de instrucciones (lenguaje) y las ubicaciones de los operandos (registros y memoria). Existen muchas arquitecturas diferentes, como RISC-V, ARM, x86, MIPS, SPARC y PowerPC. (Harris and Harris, (2019), pp. 299)

Se puede decir que la arquitectura es la *especificación*, es decir, lo que debe hacerse.

**2.2 ARQUITECTURA RISC-V**

RISC-V es una arquitectura de conjunto de instrucciones (ISA) basada en el diseño de computadoras de conjunto reducido de instrucciones (RISC, por sus siglas en inglés). Fue desarrollada inicialmente en la Universidad de California, Berkeley, y es abierta y gratuita, lo que permite su uso sin restricciones de licencias o patentes. (Harris and Harris, (2019), pp. 300)

La arquitectura RISC-V se inspiró en cuatro principios articulados por David Patterson y John Hennessy en su libro Computer Organization and Design: (1) La regularidad favorece la simplicidad, (2) Hacer los casos comunes, (3) Más pequeño es más rápido y (4) un buen diseño requiere buenos compromisos. Estos principios guían el diseño de RISC-V hacia la eficiencia, simplicidad y adaptabilidad, manteniendo un balance entre funcionalidad y facilidad de implementación. (Harris and Harris, (2019), pp. 300)

**2.2.1 CONJUNTO DE INSTRUCCIONES RV32I**

RV32I es el conjunto base de instrucciones de RISC-V para procesadores de 32 bits. Es el punto de partida para implementar la arquitectura y está diseñado para ser sencillo y eficiente.

**Características clave de RV32I:**

* Longitud fija: Todas las instrucciones tienen 32 bits, lo que simplifica la decodificación.
* Operaciones soportadas:
  + Operaciones aritméticas y lógicas (suma, resta, AND, OR, etc.).
  + Manipulación de bits (desplazamientos y rotaciones).
  + Acceso a memoria mediante operaciones de carga (load) y almacenamiento (store).
  + Control de flujo mediante saltos y llamadas condicionales o incondicionales.
* Conjunto reducido de registros: Utiliza 32 registros de propósito general, cada uno de 32 bits, incluyendo un registro dedicado al valor cero.
* Independencia: Es suficiente para ejecutar un sistema operativo básico o programas simples sin necesidad de extensiones adicionales.

Captura de pantalla de computadora

Descripción generada automáticamente

Tabla Instrucciones de números enteros de RISC-V (Harris and Harris, (2019), Instruction Set Summary)

**2.3 CONJUNTO DE REGISTROS RISC-V**

El conjunto de registros en la arquitectura RISC-V es una colección de 32 registros de propósito general, cada uno de un tamaño fijo que depende de la variante de la arquitectura: 32 bits para RV32, 64 bits para RV64 y 128 bits para RV128. Estos registros se identifican por números del x0 al x31, y también tienen nombres simbólicos que indican su propósito en convenciones comunes de programación.

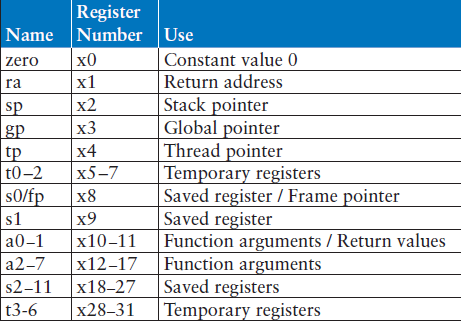


Tabla Números y nombres de registros RV32 (Harris and Harris, (2019), pp. 2)

**2.4 MICROARQUITECTURA DE COMPUTADORA**

La microarquitectura, también conocida como arquitectura de nivel de implementación, es el diseño interno y la organización de un procesador o CPU (Unidad Central de Procesamiento). Representa el puente entre la arquitectura de la computadora, que define las funciones y el comportamiento del sistema, y la implementación física del hardware.

**3 RESULTADOS**

Se puede observar en la figura 1, el diagrama final de lo que se desea obtener para realizar la unidad central de procesamientos (CPU, de las siglas en inglés, *Central Processing Unit*) desarrollándolo con la arquitectura RISC-V. Para lograr tamaña empresa, como principiantes en este tema, se ha tomado la decisión de desarrollar cada bloque de los bloques por separado para garantizar su correcto funcionamiento y facilitar su construcción. Se tomaron los algunos componentes de proyectos anteriores, por ejemplo, la memoria RAM de 256x32bits y el conjunto de registros 32x32bits que fueron desarrollados en el proyecto 9. También se tomó la Unidad Aritmética-Lógica (ALU) y todos sus componentes del proyecto 8, adicionándose el registro de 32 bits con reset. Esto quiere decir que faltaría desarrollar los siguientes bloques:

* Máquina de Estado Finito de Control: el bloque denominado MEF\_control.
* Valor inmediato: el bloque denominado valor\_inmediato
* Control de la Unidad Aritemética-Lógica: el bloque denominado control\_alu.
* Condición de salto: el bloque denominado condicion\_branch

Diagrama, Esquemático

Descripción generada automáticamente

Figura Diagrama esquemático del diseño final esperado. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.1 VALOR INMEDIATO**

Las líneas de código de este bloque se encuentran en el archivo “*valor\_inmediato.vhd*” y su simulador en el archivo “sim\_valor\_inmediato.vhd”. El componente consta de 2 entradas y 1 salida. Las cuales son:

* Entradas: Una entrada es de selección de 3 bits, que proviene del bloque MEF\_control, encargado de seleccionar el tipo de valor inmediato que se tendrá en la salida. La otra entrada es de instrucción, que tiene toma los bits 31 al 7 de los bits de instrucción del contador de programa.
* Salida: Posee una única salida, denominada *inmediato*, que es un valor de 32 bits formado de acuerdo al valor de selección.

Diagrama, Esquemático

Descripción generada automáticamente

Figura Bloque valor inmediato. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.2 CONTROL DE LA UNIDAD ARITMETICA LOGICA**

Las líneas de código de este bloque se encuentran en el archivo “*control\_alu.vhd*” y su simulador en el archivo “sim\_control\_alu.vhd”. El componente consta de 3 entradas y 1 salida. Las cuales son:

* Entradas:
  + Funct3: entrada de 3 bits, tomados de los bits 14 al 12 de los bits de instrucción.
  + Funct5\_7: entrada de 1 bit, tomado del bit 30 de los bits de instrucción.
  + Modo: señal de entrada de 2 bits proveniente de la MEF de control.
* Salida: Única salida denominada fn\_alu, de 4 bits, que determinará qué operación deberá realizar la unidad aritmeticológica (ALU).

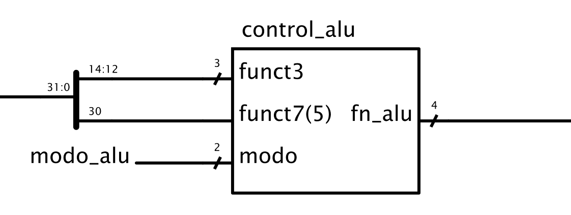


Figura Diagrama del bloque modo\_alu. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.3 CONDICION DE SALTO O BRANCH CONDITION**

Las líneas de código de este bloque se encuentran en el archivo “*branch\_condition*” y su simulador en el archivo “*sim\_branch\_condition*”. El componente consta de 1 entrada y 1 salida. Las cuales son:

* Entrada: función de 3 bits, denotado funct3, que toma los bits 14 a 12 de los bits de instrucciones.
* Salida: Salto de Z, denotado Z\_branch, que es una salida de 1 bit utilizada en la condición de salto del programa.

Este bloque es, en simples palabra, un selector. La salida será de ‘1’ cuando los bits tomados de la instrucción sean “000” o “101” o “111”. Caso contrario, la salida será ‘0’.

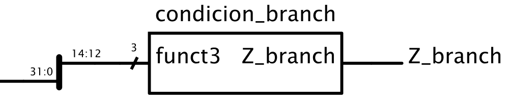


Figura Diagrama del bloque condicion\_branch. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.4 MAQUINA DE ESTADO FINITO DE CONTROL**

Las líneas de código de este bloque se encuentran en el archivo “MEF\_control” y su simulador en el archivo “sim\_MEF\_control”. El componente consta de 4 entradas y 11 salidas. Las cuales son:

* Entradas:
  + clk: entrada de un bit. Es el reloj del bloque.
  + hab\_pc: entrada de un bit. Es la habilitación del contador de programa
  + reset: entrada de un bit. Se encarga de reiniciar el programa, poniendo valores nulos en la memoria.
  + op: entrada de 7 bits tomada de los bits 6 al 0 de los bits de instrucción.
* Salidas:
  + Salidas de 1 bit:
    - esc\_pc: Escritura del contador de programa.
    - branch: salto.
    - sel\_dir: selector de dirección.
    - esc\_mem: escritura de memoria.
    - esc\_instr: escritura de instrucción.
    - Esc\_reg: escritura de registro.
  + Salidas de 2 bits:
    - Modo\_alu: selector de modo de la ALU.
    - Sel\_op1: Selector de operando 1.
    - Sel\_op2: Selector de operando 2.
    - Sel\_y: Selector de salida.
  + Salida de 3 bits:
    - sel\_inmediato: Selector del valor inmediato

Funcionamiento: En cada flanco ascendente de reloj con reset en ‘0’, lee el valor de operación dada por los bits 6 al 0 de instrucción y procede a realizar las selecciones correspondientes para todos los valores de salida. Si reset se encuentra en ‘1’, entonces todos los valores toman los valores por defecto ya predeterminados en el programa.

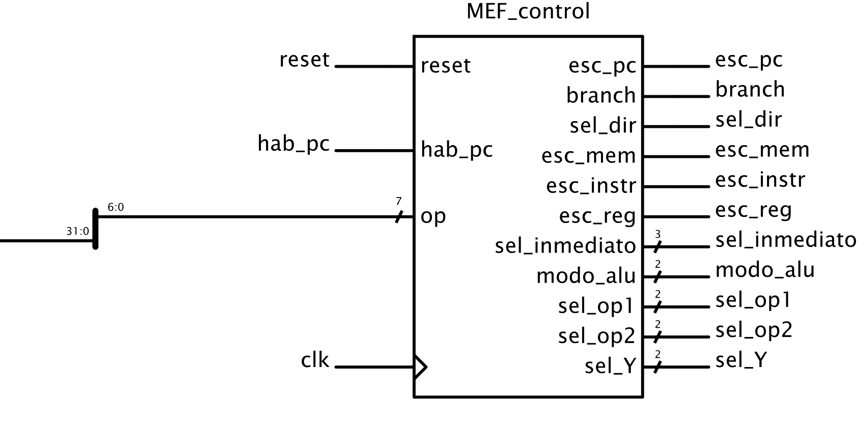


Figura Diagrama del bloque de MEF\_control. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.5 REGISTROS**

**3.5.1 REGISTRO DE 32 BITS**

Las líneas de código de este bloque se encuentran en el archivo “*registro32.vhd*” y su simulador en el archivo “*sim\_registro32.vhd*”. El componente consta de 4 entradas y 1 salidas. Las cuales son:

* Entradas:
  + Reset: entrada de un bit. Encargado de establecer en “0” los bits de salida Q.
  + Hab: entrada de habilitación de 1 bit.
  + D: entrada de datos de 32 bits.
  + Clk: reloj del programa.
* Salidas: Salida de dato de 32 bits, denotado Q.

Funcionamiento: Cuando reset sea ‘0’ y la habilitación esté en ‘1’, pasará los datos en D a Q en el flanco ascendente del reloj (clk). Si reset es igual a 0 durante el flanco ascendente del reloj, pondrá la salida Q en “0” sin importar el valor de la habilitación o el valor en D.

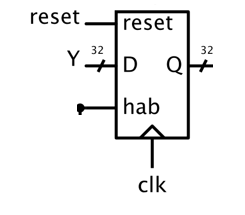


Figura Diagrama en bloque de un registro de 32 bits. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.5.2 CONJUNTO DE REGISTROS 32X32 BITS**

Las líneas de código de este bloque se encuentran en el archivo “*reg\_32x32*” y su simulador en el archivo “*sim\_reg\_32x32*”. El componente consta de x entradas y x salidas. Las cuales son:

* Entradas:
  + Dir\_1: dirección 1, consta de 5 bits tomados de los bits 19 a 15 de los bits de instrucción.
  + Dir\_2: dirección 2, consta de 5 bits tomados de los bits 24 a 20 de los bits de instrucción.
  + Dir\_escritura: dirección de escritura, consta de 5 bits tomados de los bits 11 a 7 de los bits de instrucción.
  + Dat\_escritura: señal de datos de escritura de 32 bits, proveniente de la señal de salida Y.
  + Hab\_escritura: habilitación de escritura, señal de 1 bit proveniente de la máquina de estado finito de control.
  + Clk: Reloj de programa. Es una señal de 1 bit.
* Salidas:
  + Dat\_1: señal de datos de salida de 32 bits.
  + Dat\_2: señal de datos de salida de 32 bits.

Funcionamiento: Escribe el valor Y en la dirección de memoria cuando está habilitada la escritura (esc\_reg=’1’) durante el flanco ascendente del reloj (clk).

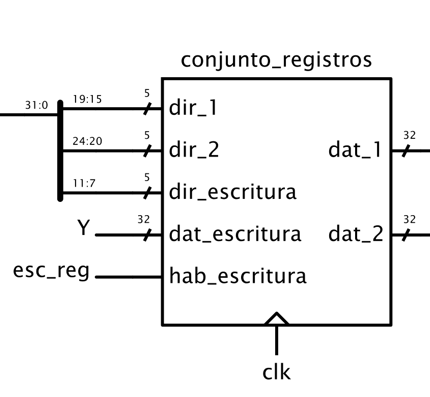


Figura Diagrama de un conjunto de registros 32x32bits. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**3.8 MEMORIA RAM**

Las líneas de código de este bloque se encuentran en el archivo “*ram\_256x32.vhd*” y su simulador en el archivo “s*im\_ram\_256x32.vhd*”. El componente consta de 7 entradas y 1 salidas, de las que se usarán 4 entradas: el puerto de dirección (dir), la habilitación de escritura (hab\_w), datos de escritura (dat\_w) y el reloj (clk); además de la única salida (dat\_r).

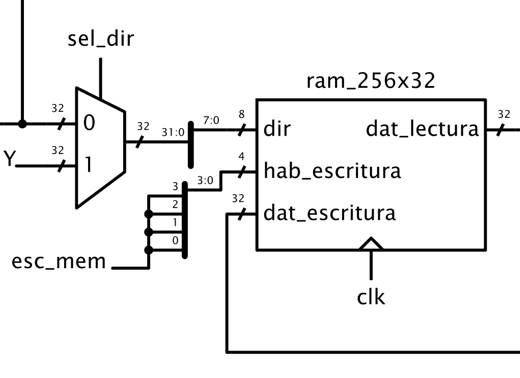
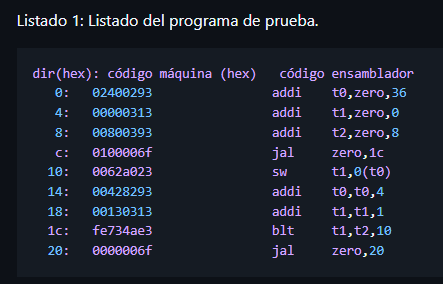
****

Figura 8 Diagrama en bloque de una ram\_256x32. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

**4 EJECUCIÓN DEL PROGRAMA**

Luego de completar los distintos componentes, se ha procedido a conectarlos en un programa nombrado *cpu.vhd*, que en conjunto con su simulador nombrado *sim\_cpu.vhd*, deberá correr el siguiente listado de instrucciones:



Listado Listado del programa de prueba. Adaptado de “Proyecto 10 - Núcleo Risc-V, primera versión”, por F. Bonomi, 2024, GitHub (https://github.com/fmirandabonomi/eii\_proyecto\_10.git)

El listado de la imagen se guarda en un archivo nombrado origen.mem, el cual será leído desde el simulador del CPU conectado a una memoria RAM de 256x32bits. Las instrucciones se escribirán en las primeras 9 posiciones de la memoria, es decir, de la posición 0 hasta la posición 8. Luego, de las posiciones 9 a 16 se escriben los resultados de las instrucciones, que en este caso serán los valores de 0 a 7, empezando con el valor 0 en la posición 9 hasta el valor 7 en la posición 16.

A continuación se muestra el resultado de la simulación:

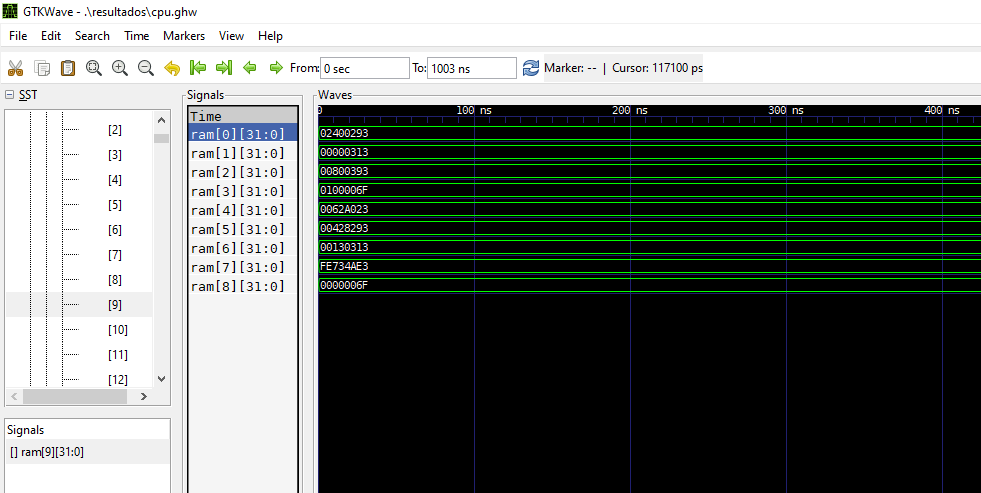


Figura Imagen del programa GTKWave de las direcciones de memoria 0 al 8 en donde se encuentra escritas las intrucciones de programa.

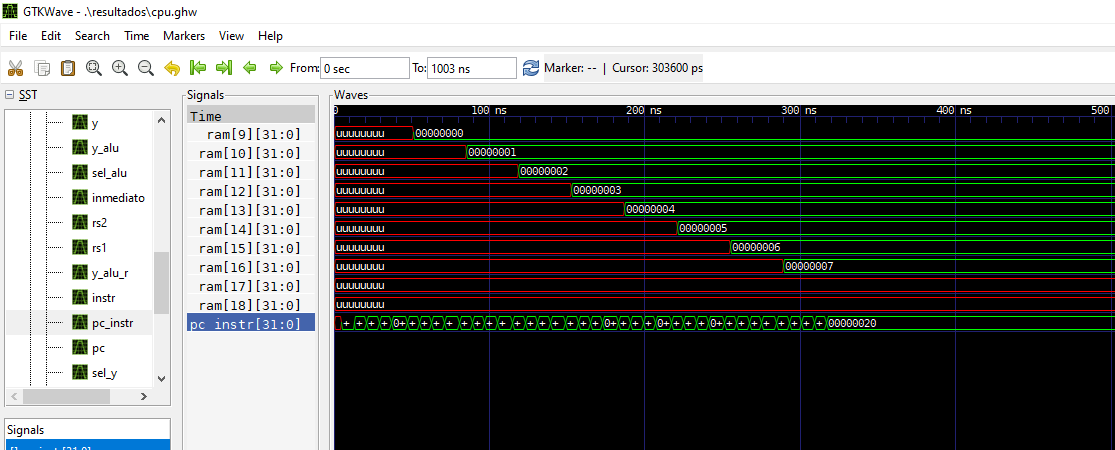


Figura Imagen de GTKWave del contenido de las posiciones 9 a 18 de la memoria RAM

Con estos resultados queda demostrado que la ejecución del programa es correcta.

**5 CONCLUSIONES**

La arquitectura de computadora define el "qué" de un sistema de cómputo, estableciendo cómo se diseñan y organizan los componentes para realizar tareas computacionales. Su comprensión es esencial para optimizar el rendimiento, garantizar compatibilidad y desarrollar tecnologías en la computación moderna.

La microarquitectura es un nivel esencial en el diseño de computadoras que traduce la lógica abstracta de la arquitectura a una implementación física eficiente y funcional. Su análisis y optimización son fundamentales para avanzar en el rendimiento y la eficiencia de los procesadores modernos.

En conclusión, la arquitectura de computadoras, la microarquitectura y la arquitectura RISC-V son elementos complementarios que trabajan en distintos niveles del diseño de sistemas computacionales. Mientras que la arquitectura proporciona una base teórica y funcional, la microarquitectura traduce esas ideas en hardware eficiente. RISC-V, con su enfoque abierto y modular, marca un hito en la evolución de las arquitecturas, ofreciendo flexibilidad y acceso sin precedentes, lo que promete revolucionar el diseño de procesadores en las próximas décadas.

**6 REFERENCIAS**

Harris, S. and Harris D. (2019). *Digital Design and Computer Architecture, RISC-V Edition.* Editorial Morgan Kaufmann.

John F. Wakerly. (2006). *Digital Design: Principles and Practices*. Prentice Hall.