**TG-1937**

DISEÑO DE UN PROTOTIPO DE CONTROLADOR PID EN DRONES CON PROCESADOR A LA MEDIDA EN ARQUITECTURA RISC-V PARA IMPLEMENTACIÓN EN FPGA

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

DEPARTAMENTO DE ELECTRÓNICA

BOGOTÁ, D.C.

2020

DISEÑO DE UN PROTOTIPO DE CONTROLADOR PID EN DRONES CON PROCESADOR A LA MEDIDA EN ARQUITECTURA RISC-V PARA IMPLEMENTACIÓN EN FPGA

**Autores:**

Iván Ricardo Diaz Gamarra

Omar Steck Espinel Santamaria

Magda Daniela Latorre Ortiz

MEMORIA DE TRABAJO DE GRADO PARA OPTAR EL TITULO DE INGENIERO ELECTRONICO

**Director**

Ing. Francisco F. Viveros, Profesor Emérito.

**Codirectores**

Ing. Luisa F. García, M.Sc.

Ing. Jose L. Uribe, M.Sc., MBA.

**Juries of the Undergraduate Final Project**

<Name of the jury >

< Name of the jury >

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

DEPARTAMENTO DE ELECTRÓNICA

BOGOTÁ, D.C.

JUNIO, 2020

**PONTIFICIA UNIVERSIDAD JAVERIANA**

**FACULTAD DE INGENIERIA**

**DEPARTAMENTO DE ELECTRÓNICA**

**Rector de la Pontificia Universidad Javeriana**

Jorge Humberto Peláez Piedrahita, S.J.

**Decano de la Facultad de Ingeniería**

Ing. Lope Hugo Barrero Solano

**Director de la Carrera de Ingeniería Electrónica**

Ing. Alejandra María González Correal

**Director del Departamento de Electrónica**

Ing. Diego Alejandro Patiño Guevara

**Artículo 23 de la Resolución No. 1 de junio de 1946**

“La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia”

**AGRADECIMIENTOS**

*Mensaje de Ivan*

*Iván Diaz*

*Mensaje de Omar*

*Omar Espinel*

*Mensaje de Daniela*

*Daniela Latorre*

**CONTENT**

[I- INTRODUCCIÓN 1](#_Toc467749631)

[II- DESCRIPCIÓN GENERAL 2](#_Toc467749632)

[1. Opportunity, Problem 2](#_Toc467749633)

[1.1. Problem Context 2](#_Toc467749634)

[1.2. Problem Formulation 2](#_Toc467749635)

[1.3. Solution Proposal 2](#_Toc467749636)

[1.4. Solution Justification 2](#_Toc467749637)

[2. Project Description 2](#_Toc467749638)

[2.1. General Objective 3](#_Toc467749639)

[2.2. Specific Objectives 3](#_Toc467749640)

[2.3. Deliverables, Standards, and Justification 3](#_Toc467749641)

[III- CONTEXTO 3](#_Toc467749642)

[1. Background 3](#_Toc467749643)

[2. Context Analysis 3](#_Toc467749644)

[IV- ANÁLISIS DEL PROBLEMA 3](#_Toc467749645)

[1. Requirements 4](#_Toc467749646)

[2. Constraints 4](#_Toc467749647)

[3. Functional specification 4](#_Toc467749648)

[V- DISEÑO 4](#_Toc467749649)

[VI- IMPLEMENTACIÓN 4](#_Toc467749650)

[VII- RESULTADOS 4](#_Toc467749651)

[VIII- CONCLUSIONES 5](#_Toc467749652)

[1. Impact Analysis of the Project 5](#_Toc467749653)

[2. Conclusions and Future Work 5](#_Toc467749654)

[IX- REFERENCIAS 5](#_Toc467749655)

[X- APENDICES 5](#_Toc467749656)

**ABSTRACT**

**RESUMEN**

1. INTRODUCCIÓN

El presente documento trata el contexto actual sobre los procesadores libres a la medida y sus aplicaciones en drones y controladores, así como muestra el desarrollo de un prototipo de procesador basado en la arquitectura RISC-V. La estructura del documento está pensada de forma tal que el lector pueda familiarizarse con el contexto de los procesadores libres, comprenda los términos fundamentales, para luego dar lugar al proceso de ingeniería en el desarrollo del prototipo de forma detallada.

A lo largo del proyecto, se investiga sobre el estándar RISC-V y se muestra el uso de un proceso de ingeniería para generar a partir de este un procesador especializado en el calculo de algoritmo PID para control de vuelo de drones, creando una ALU enfocada en la velocidad, en especial en las operaciones de suma y multiplicación, se muestra adicionalmente el diseño de los distintos componentes que integran el procesador y su posterior implementación en lenguaje de descripción de hardware.

El documento inicia identificando una potencial área de investigación y desarrollo para procesadores libres en el área de los controladores para drones. El capítulo del contexto del proyecto expone los conceptos más importantes para la comprensión del proyecto y del documento, así como la comparación con otros proyectos similares en el campo de procesadores libres. Por otra parte, en el capítulo de análisis del problema presenta los requerimientos, restricciones y una especificación del prototipo que es ampliada posteriormente en los capítulos de diseño e implementación. Para comprobar el funcionamiento del prototipo y validar la arquitectura planteada se muestra una sección de pruebas donde se encuentran los resultados de una implementación física del proyecto.

Finalmente, se encuentran conclusiones del equipo sobre los resultados del proyecto y un análisis de resultados, junto a una sección de trabajo futuro para la implementación del prototipo.

1. DESCRIPCIÓN GENERAL

## Oportunidad, Problema y Antecedentes

### Contexto del problema

Los procesadores juegan un rol fundamental en el mundo de la tecnología y los elementos digitales, interpretando y ejecutando instrucciones que definen el comportamiento completo de un sistema.

Existe una gran variedad de procesadores, con múltiples funcionalidades, aplicaciones, características e instrucciones. Esta variedad en los procesadores está dada por las diferencias en la arquitectura, es decir la forma en la que están construidos, que componentes los integran y que funciones cumplen[1]. Cada fabricante tiene sus propias arquitecturas, encontrando diferentes componentes, instrucciones y funcionalidades en procesadores con el mismo propósito, estas diferencias de arquitecturas entre todos los procesadores y el hecho de que estas sean registradas y no puedan ser usadas ni estudiadas por otros, son un impedimento para encontrar un estándar base para la arquitectura de estos.

Una solución para lograr estandarizar una arquitectura de procesadores se presenta en el área del open hardware, donde existen iniciativas de arquitecturas libres, como MIPS[2] y RISC-V[3], que buscan plantear un estándar para las arquitecturas de procesadores. Estas arquitecturas, pueden ser consultadas, e implementadas por cualquiera, desde académicos hasta empresas, permitiendo el uso, investigación y aplicación de estas arquitecturas, a diferentes áreas de la tecnología.

Una de las áreas de la tecnología que hace amplio uso de los procesadores, es el área de los *drones*, donde se usan procesadores para el control del vehículo, en específico en un área denominada, el controlador de vuelo, donde se usa un procesador que computa diferentes señales para lograr el vuelo del drone. Los procesadores usados en esta área, como el Pixhawk[4], GAP8[5] o Qualcomm Flight Pro[6]. Son creados por compañías privadas que incluso aunque algunos implementan arquitecturas libres. La arquitectura propia de estos procesadores es cerrada y no esta disponible para ser consultada, estudiada o implementada por particulares, por lo que el estudio sobre las arquitecturas en este tipo de aplicación es limitado y no existe material que detalle el proceso de arquitectura de un procesador con las instrucciones para este tipo de aplicaciones.

### Formulación del problema

El problema encontrado, radica en la ausencia de una arquitectura , que sea libre, documentada y abierta, en el ámbito de los procesadores usados en controladores de vuelo para drones, ya que las arquitecturas de los procesadores disponibles en esta área no permiten estudiar la implementación de procesadores para propósitos específicos desde su arquitectura, así como tampoco permiten conocer la implementación de los estándares de arquitecturas abiertas en esta área.

### Propuesta de la solución

Para solucionar el problema se propone la creación del prototipo de un procesador a la medida, basado en una arquitectura libre, para esta área en específico, que pueda ser implementado para su uso y adicionalmente, pueda ser estudiado y desarrollado de forma incremental, de acuerdo a los permisos otorgados por la arquitectura seleccionada y el tipo de licencia sobre la cual se publica el prototipo desarrollado.

Para la arquitectura del procesador, la solución implementa la arquitectura RISC-V y se genera como entregable la documentación del proceso de diseño y desarrollo, que permite el estudio a fondo de la arquitectura, su composición y decisiones de diseño. Esto permite abrir el camino a una nueva rama de aplicación documentada para el uso de un estándar de arquitectura libre.

El prototipo propuesto, debe cumplir con las operaciones de un procesador de propósito general, pero con el objetivo de delimitar el proyecto para ajustarse a los plazos establecidos el prototipo diseñado, se enfoca en uno de los elementos fundamentales que debe realizar el procesador para control de vuelo, que es la optimización en tiempo de las operaciones requeridas para el algoritmo de control del drone.

### Justificación de la solución

Para solucionar la falta de una arquitectura de procesador libre en el área seleccionada, se considera oportuno como solución proveer un prototipo de procesador y la completa documentación de la aplicación de una arquitectura en este proyecto.

Adicionalmente, se propone el uso de la arquitectura RISC-V, una arquitectura libre y gratuita para el conjunto de instrucciones de procesadores, que tiene la ventaja de contar con el respaldo de 325 compañías, siendo una arquitectura que sigue un estándar y la facilidad de contar con varios ejemplos e implementaciones con código libre para su estudio, lo que representa una mayor cantidad de información relacionada al tema[7], favoreciendo al desarrollo del proyecto.

El desarrollo de un procesador a la medida para el controlador de vuelo de drones permite enfocar el desarrollo del hardware, en este caso del procesador, en torno a una aplicación específica. Y debido a la complejidad que representa la solución, por las características que debe cumplir un procesador de este tipo en cuanto a: tiempos de respuesta, cómputo y manejo de periférico [8] [9], se delimita el proyecto, con el objetivo de hacerlo alcanzable dentro del tiempo propuesto, limitándose, a un procesador a la medida enfocado al controlador PID usado por un dron. Reduciendo de esta manera en el procesador propuesto el tiempo de ejecución de los cálculos para el controlador PID.

La selección del método de control se realiza bajo el criterio de ¿cuál es el método más usado actualmente en drones? De acuerdo a Chen et al.[10] y a Chao et al.[11], el más usado en drones, es el control por PID [12], por lo que se busca optimizar la respuesta del procesador para una operación que se realiza continuamente mientras el dron se encuentra en el aire.

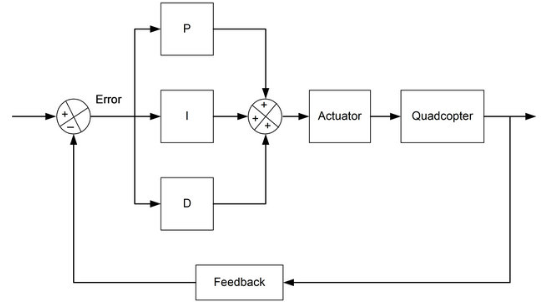


Figura. 1 control PID de un dron Quadcopter [13]

Este proyecto reporta un beneficio significativo para el sector de los drones al proveer de una arquitectura diseñada específicamente para el controlador del dron, presentando una alternativa open hardware a los procesadores usados actualmente, estos resultados, no solamente se presentan como un avance en investigación sino como una aplicación, que puede ser usada posteriormente para el desarrollo comercial.

Finalmente, este proyecto representa beneficios para los ingenieros y comunidad interesada en el hardware libre, específicamente en las distintas aplicaciones del RISC-V, proporcionando información y una guía detallada del desarrollo orientado a una aplicación en el área de drones controlados.

## Descripcion del proyecto

### Objetivo General

Diseñar el prototipo de un procesador a la medida para el controlador PID en drones, basándose en la arquitectura de conjunto de instrucciones reducido RISC-V.

### Objetivos Especificos

* Seleccionar las instrucciones deseadas de la arquitectura RISC-V para el procesador.
* Describir el sistema en un lenguaje de transferencia de datos AHPL.
* Diseñar el procesador para la aplicación seleccionada.
* Implementar un prototipo del sistema diseñado para realizar las pruebas sobre el procesador.
* Implementar un sistema complementario de pruebas que emule las entradas de los procesadores y el microcontrolador, para realizar las pruebas de tiempo de ejecución del cálculo para el control PID.
* Comparar el tiempo de ejecución para el mismo cálculo de control PID en el procesador con respecto a un procesador y a un micro-controlador usados en drones en la actualidad.

### Metodología

Para la parte de diseño del proyecto de grado, se ha escogido una metodología de diseño top down, esta metodología es usada en proyectos de desarrollo de circuitos, la cual consiste en primero descomponer el problema o el proyecto en módulos que resuelvan las diferentes necesidades de manera independiente y realizar de esta manera el diseño desde una perspectiva generalizada[14], a medida que se avanza en etapas posteriores del diseño las descripciones de los componente o módulos se hacen más específicas y se toman las decisiones de diseño pertinentes hasta llegar a una implementación que cumpla con los requisitos propuestos.

La primera fase del proyecto es la fase de levantamiento de requisitos, se realiza la investigación tanto de las operaciones que realiza el procesador, como de las instrucciones que provee la arquitectura RISC-V con este proceso se identifican las especificaciones del sistema y las características de los bloques con base al estándar de la arquitectura seleccionada.

La siguiente fase del trabajo es la fase de diseño donde se asignan las funcionalidades a los bloques que integran el sistema y se delimitan estos bloques con la ayuda de las características identificadas previamente, este proceso viene acompañado de la creación de un diagrama de bloques, se describen los bloques y se diseñan los elementos que los componen. Posteriormente, se realiza, la descripción del diseño mediante un lenguaje de transferencia de registros como el AHPL que describe el funcionamiento del sistema.

La siguiente fase es la fase de desarrollo, a partir del diseño elaborado en la fase anterior se identifican los bloques a diseñar y se usa el lenguaje de especificación VHDL para describir bloques lógicos funcionales que cumplan con lo especificado en la fase anterior, en esta fase se realiza una descripción final del diseño de cada bloque y las decisiones finales del diseño

Finalmente se tiene la etapa de pruebas donde el procesador configurado en la FPGA es probado para verificar que cumpla exitosamente con las instrucciones escogidas.

1. CONTEXTO

## Marco teórico

El elemento central del proyecto es un procesador, que se define como un dispositivo que procesa los datos, controla el funcionamiento de los dispositivos y se encarga de realizar las operaciones lógicas y matemáticas.[15] A partir de esto, se puede obtener una idea de la importancia del procesador en un sistema digital, ya que es el encargado de controlar el comportamiento de los diferentes elementos, y procesar todas las instrucciones necesarias para hacer funcionar el sistema, las instrucciones que ejecuta el procesador están dadas por la arquitectura del conjunto de instrucciones o por sus siglas en ingles ISA (Instruction Set Architecture) que es la interfaz entre el programador y el procesador, siendo una lista de instrucciones que puede llevar a cabo el procesador al ingresar ciertos datos[16], pero no es todo, el conjunto de instrucciones, contiene también los modos de direccionamiento, los tipos de dato, registros y definiciones generales sobre la arquitectura del procesador.

Estas instrucciones y la información que ofrece del sistema, es un elemento fundamental para el procesador, ya que los elementos internos de este, su unión, y la forma en que interactúan es lo que se llama la arquitectura del procesador.[1]

Las arquitecturas pueden ser de dos tipos: arquitecturas libres, que están abiertas y disponibles al publico para su estudio, haciendo el añadir, mejorar e intercambiar componentes más fácilmente.[17] y las arquitecturas cerradas, que son sistemas con especificaciones que no son públicas y prohíben que terceros creen, mejoren o añadan elementos usando esta arquitectura.[18]

En cuanto a arquitecturas libres existen algunas opciones como MIPS [2] y RISC-V [3], que proveen especificaciones arquitecturales en procesadores de diferentes tipos y para múltiples propósitos, siendo la última sumamente relevante al ser una arquitectura simple, reciente, en constante crecimiento y con un amplio apoyo tanto de comunidad y de empresas que usan la arquitectura como un estándar para la creación de procesadores innovando en diferentes áreas y siendo aplicado en numerosos dispositivos actuales. [7]

La arquitectura RISC-V esta compuesta por un ISA detallado, que define varios conjuntos de instrucciones según las necesidades del procesador y que permite a partir del conjunto de instrucciones implementadas, el diseño de los distintos elementos que componen un procesador, decidiendo los tamaños de memoria, los bloques que conforman el procesador y las operaciones que lleva la ALU, (Unidad aritmeticológica, por sus siglas en ingles Arithmetic Logic Unit), siendo esta uno de los elementos más importantes dentro de un procesador, ya que es en esta donde se realizan las operaciones lógicas y aritméticas, que permiten generar los datos de las instrucciones a las salidas, para realizar las instrucciones que ejecuta el procesador[19].

Las operaciones que realiza la ALU van desde operaciones sencillas, como operaciones lógicas entre bits: AND, OR, XOR, hasta operaciones mas complicadas con tipos de datos especiales como la multiplicación o división de números flotantes.

## Estado del arte

A continuación, se muestra una línea de tiempo donde se muestran los avances más significativos en los últimos 15 años con respecto a la arquitectura libre RISC-V, la implementación de procesadores con esta arquitectura y el desarrollo de hardware a la medida enfocado a drones: [20] [21] [22] [6] [23] [24] [25] [26]



Figura. Estado del arte del proyecto

1. ANÁLISIS DEL PROBLEMA

## Requerimientos

Para la toma de requerimientos de la solución, el equipo, bajo el rol de dueño del problema se reunió para decidir cuáles son las características con las que debe contar la solución propuesta. Una vez identificadas estas características se escriben en forma de requerimientos y se obtienen los siguientes:

* El sistema debe ser un procesador que implemente las instrucciones básicas de un procesador de propósito general.
* El procesador debe tener palabras de 32 bits.
* El procesador debe cumplir con los estándares y características de la arquitectura Risc-V.
* El sistema debe estar especializado en las operaciones que se requieren para el calculo del algoritmo de control PID: Suma y Multiplicación.
* La complejidad del procesador debe ser suficiente para ser desarrollado en un plazo no mayor a 4 meses.
* El procesador debe contar con salidas que permitan la visualización de señales internas de importancia con el fin de comprobar su correcto funcionamiento.
* El procesador debe ser implementado sobre una FPGA.

A partir de estos requerimientos se deciden las funcionalidades del proyecto.

## Restricciones

A continuación, se presentan las restricciones sobre el sistema, es decir, aquellas que restringen la solución:

El tamaño del sistema debe ser suficiente para ser almacenado en una FPGA.

La cantidad de entradas y salidas del sistema esta limitada por la cantidad de pines de entrada/salida de la FPGA.

Diseño de la arquitectura del sistema.

Los formatos de instrucción del estándar RISC-V.

Las limitaciones de la arquitectura RISC-V.

Limitaciones del lenguaje de descripción VHDL.

Las pruebas están limitadas por la cantidad de señales que pueden ser emitidas por el sistema y las entradas del analizador lógico.

Limitaciones del tiempo de desarrollo.

## Especificación funcional

La especificación funcional describe las funciones que debe realizar el sistema, a continuación, se listan y se describen las funcionalidades principales que implementa el sistema:

Leer instrucciones: el sistema lee instrucciones de una dirección de memoria determinada y almacenada en un registro,

Ejecutar instrucción: el sistema al leer las instrucciones realiza acciones de acuerdo con los datos contenidos en esta.

Leer y escribir en memoria y registros: El sistema puede almacenar o leer datos, de un conjunto de registros internos, así como de un bloque de memoria externo.

Atender periféricos: El sistema permite interrupciones por parte de los periféricos y atiende dichas interrupciones.

Instrucciones de propósito general: El sistema posee instrucciones de un procesador de propósito general.

Operaciones aritmeticológicas: El sistema posee una ALU que permite realizar operaciones aritmeticológicas.

1. DISEÑO

# Instrucciones.

## Instrucciones necesarias.

Para comprender cuales instrucciones eran necesarias, se consideraron dos elementos fundamentales: ¿Qué instrucciones necesita el sistema? ¿Qué instrucciones provee el estándar RISC-V?

Para encontrar las instrucciones que necesita el sistema se toma como referencias dos de los los procesadores usados en control para drones: El ARM Cortex M4[27], procesador de la SMT 32 y el Atmel 7810[28], procesador del Arduino uno.

Al analizar las instrucciones de estos dos procesadores, se crea una lista de las instrucciones comunes entre estos dos procesadores y se seleccionan las instrucciones fundamentales para implementar en el sistema:

* Move
* Add
* Substract
* Multiply
* Compare
* AND
* XOR
* OR
* NOT
* Shift
  + Shift lógico
  + Shift aritmético
* Load
* Store
* Push
* Pop
* Jump

## Instrucciones en RISC-V

Las instrucciones seleccionadas se buscan en el estándar RISC-V, al consultar [29]se encuentran los diferentes conjuntos de instrucciones del estándar, existen múltiples variaciones del estándar RISC-V, todas ellas empiezan por las letras RV y continúan, de acuerdo con el conjunto de instrucciones implementado, a continuación, se muestran y explican las principales:

RV32I y RV64I: Conjunto de instrucciones base del estándar RISCV, para 32 y 64 bits, contiene las instrucciones generales de manejo de registros, operaciones aritmeticológicas con enteros, así como de memoria y son implementadas en el resto de los estándares con excepción del conjunto de instrucciones comprimido.

RV32M y RV64M: Extensión del conjunto de instrucciones base para la multiplicación, en 32 y 64 bits.

RV32FD y RV64D: Extensión del conjunto de instrucciones base para la división, en 32 y 64 bits.

RV32V y RV64V: Extensión del conjunto de instrucciones base para operaciones aritmeticológicas entre vectores.

RV32C :Conjunto de instrucciones comprimidas, extensión que resume varias instrucciones del estándar RV32I en equivalentes de 16 bits.

Después de consultar el estándar RISC-V se usa el conjunto de instrucciones RV32I, ya que es la base del estándar que debe ser implementada en todos los sistemas que incluyan el estándar.

Una de las ventajas que ofrece el conjunto RV32I para el sistema, radica en que incluye la mayoría de las instrucciones seleccionadas para implementa, ante las instrucciones faltantes, se decide expandir mediante el módulo de multiplicación ya que se necesitan las multiplicaciones para la implementación, así mismo se considera la implementación de operaciones comprimidas para las operaciones del tipo push y pop, ya que si bien es posible en el estándar RV32I hacer operaciones en el stack pointer, no existen instrucciones dedicadas para push y pop.

Con el conjunto de instrucciones seleccionado, se tendrían todas las necesarias según lo consultado con los otros dos procesadores. Cumpliendo con los requisitos en materia de instrucciones.

## Formatos de instrucción

Los diferentes conjuntos de instrucciones del estándar RISC-V poseen varios formatos de instrucción, que dependen de la longitud de palabra, 32, 64 e incluso 16 para las instrucciones comprimidas.

En este caso, al implementar el estándar RV32I y la extensión de multiplicación, los formatos de instrucción usados en el procesador son los siguientes:

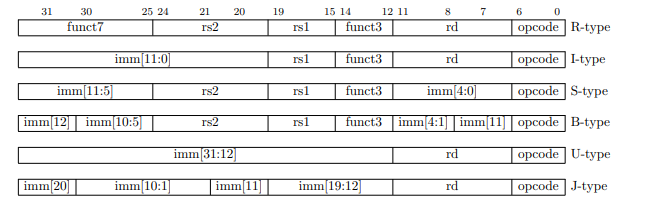


Figura. 3 Formatos de instrucccion RV32I

Los formatos de instrucción no son asignados de acuerdo con el tipo de instrucción, como operaciones en la ALU u operaciones sobre registros, sino sobre la operación especifica que se requiera para esa instrucción, el estándar usado, distribuye el espacio dentro de la instrucción de la forma mas optima para se implementado, manteniendo direcciones, datos y códigos de operación organizados y en medida de lo posible con los mismos tamaños y en el mismo espacio dentro de la instrucción.

Un elemento del formato de instrucción que es útil para la implementación es la identificación de los segmentos comunes, entre los cuales se tiene:

Código de operación: Primera parte del código de operación almacenada en los bits 0 al 6 de la instrucción, la segunda parte del código de operación, marcada como funct3, esta almacenada dentro de los bits 12 al 14.

Direcciones: Los formatos de instrucción, que contienen direcciones, almacenan las instrucciones en el mismo espacio del formato de instrucción, la dirección rd, desde el bit 7 al 11 de la instrucción, mientras que la dirección de rs1 y rs2 están almacenados desde los bits 15 al 19 para rs1, y 20 al 24 para rs2.

En el caso de las instrucciones de PUSH y POP tomadas de la extensión comprimida, el formato de instrucción cambia, a los vistos anteriormente, en este caso ambas instrucciones usan un mismo formato de instrucción, que se presenta a continuación:



Figura. 4 Formato instrucción comprimido PUSH y POP

En este formato, el opcode ocupa dos bits, desde el 0 al 1, mientras la segunda parte, el funct3, va desde el bit 13 al bit 15. La dirección del registro, en este caso rs2, se almacena del bit 2 al bit 6.

## Instrucciones a implementar

Decidido el conjunto de instrucciones y las extensiones para usar, se listan las instrucciones a ser implementadas, junto con una breve descripción de cada una, para una descripción más técnica y completa se debe consultar el manual anexo:

|  |  |
| --- | --- |
| Nombre | Descripción |
| **LUI** | Escribe un el registro el dato almacenado en *immediate*, desplazado 12 bits a la izquierda y rellenado con ceros. |
| **AUIPC** | Escribe un registro*,* la suma del pc más el dato almacenado en *immediate*, desplazado 12 bits a la izquierda y rellenado con ceros |
| **JAL** | Escribe un registro con la dirección *rd,* la dirección de la próxima instrucción. Luego suma al PC un *offset* extendido en signo. |
| **JALR** | Escribe en un registro*,* la dirección de la próxima instrucción. Luego almacena en PC la suma de offset más un registro. |
| **BEQ** | Compara dos registros, si son iguales, suma al pc el *offset* en signo extendido. |
| **BNE** | Compara dos registros, si son diferentes, suma al pc el *offset* en signo extendido. |
| **BLT** | Compara dos registros, si el primero es menor al segundo, suma al pc el *offset* en signo extendido. |
| **BGE** | Compara dos registros, si el primero es mayor al segundo, suma al pc el *offset* en signo extendido. |
| **BLTU** | Compara dos registros, como números sin signo, si el primero es menor al segundo, suma al pc el *offset* en signo extendido. |
| **BGEU** | Compara dos registros, como números sin signo, si el primero es mayor al segundo, suma al pc el *offset* en signo extendido. |
| **LB** | Carga un byte de memoria extendido en signo. |
| **LH** | Carga dos bytes de memoria extendidos en signo. |
| **LW** | Carga cuatro bytes de memoria. |
| **LBU** | Carga un byte de memoria extendido con ceros. |
| **LHU** | Carga dos bytes de memoria extendidos con ceros. |
| **SB** | Guarda en memoria el byte menos significativo de un registro, en la dirección de la suma de un registro más el offset. |
| **SH** | Guarda en memoria los dos bytes menos significativos de un registro, en la dirección de la suma de un registro más el offset. |
| **SW** | Guarda en memoria un registro, en la dirección de la suma de un registro más el offset. |
| **ADDI** | Suma un registro con el dato immediate extendido en signo. |
| **SLTI** | Compara un registro con dirección con el dato immediate extendido en signo. Si el el registro es menor, escribe 1 en un registro, de lo contrario escribe 0 en este registro. |
| **SLTIU** | Compara un registro con dirección con el dato immediate, como números sin signo. Si el registro es menor, escribe 1 en un registro, de lo contrario escribe 0 en este registro. |
| **XORI** | Escribe en un registro el XOR a nivel de bits entre otro registro y el dato immediate extendido en signo. |
| **ORI** | Escribe en un registro el OR a nivel de bits entre otro registro y el dato immediate extendido en signo. |
| **ANDI** | Escribe en un registro el AND a nivel de bits entre otro registro y el dato immediate extendido en signo. |
| **SLLI** | Mueve el contenido de un registro a la izquierda, reemplazando con ceros, el número de bits almacenado en la instrucción. |
| **SRLI** | Mueve el contenido de un registro a la derecha, reemplazando con ceros, el número de bits almacenado en la instrucción. |
| **SRAI** | Mueve el contenido de un registro a la derecha, reemplazando con el MSB, el número de bits almacenado en la instrucción. |
| **ADD** | Almacena en un registro la suma de otros dos registros. |
| **SUB** | Almacena en un registro la resta de otros dos registros. |
| **SLL** | Mueve el contenido de un registro a la izquierda, llenando con 0, la cantidad de veces almacenada en otro registro. |
| **SLT** | Compara el contenido de dos registros. Si el contenido del primero es menor, escribe 1 en otro registro , de lo contrario escribe 0. |
| **SLTU** | Compara el contenido de dos registros, como números sin signo. Si el contenido del primero es menor, escribe 1 en otro registro , de lo contrario escribe 0. |
| **XOR** | Escribe en un registro el XOR a nivel de bits entre otros dos registros. |
| **SRL** | Mueve un registro a la derecha, la cantidad de veces almacenada en otro registro, llenando los bits con 0, el resultado es escrito en un tercer registro. |
| **SRA** | Mueve un registro a la derecha, la cantidad de veces almacenada en otro registro, llenando los bits con el MSB, el resultado es escrito en un tercer registro. |
| **OR** | Escribe en un registro el OR a nivel de bits entre otros dos registros. |
| **AND** | Escribe en un registro el AND a nivel de bits entre otros dos registros. |
| **CSRRW** | Primero escribe el contenido de un registro CSR(bloque de registros especial) en un registro, luego escribe el contenido de otro registro en el mismo registro CSR. |
| **CSRRS** | Primero escribe el contenido de un registro en un registro, luego escribe el OR entre el contenido de ambos registros en el mismo registro CSR. |
| **CSRRC** | Primero escribe el contenido de un registro en un registro, luego escribe el AND entre el contenido de ambos registros en el mismo registro CSR. |
| **CSRRWI** | Primero escribe el contenido de un registro en un registro, luego escribe el dato almacenado en la instrucción en el mismo registro CSR. |
| **CSRRSI** | Primero escribe el contenido de un registro en un registro, luego escribe en el mismo registro CSR el OR entre el registro y un dato almacenado en la instrucción. |
| **CSRRCI** | Primero escribe el contenido de un registro en un registro, luego escribe en el mismo registro CSR el AND entre el registro y un dato almacenado en la instrucción. |
| **MUL** | Almacena en un registro los primeros 32 bits de la multiplicación de dos registros. |
| **SWSP** | Almacena una palabra en la memoria en la dirección del Stack pointer |
| **LWSP** | Carga en un registro, una palabra desde la memoria con dirección del Stack Pointer |

Tabla 1 Lista completa de instrucciones

## Diagrama de bloques

En la figura a continuación se muestra el diagrama en bloques propuesto para el prototipo, estos bloques fueron obtenidos y conectados a medida que se analizan las instrucciones y se desarrollaba la arquitectura, la notación para leer las señales que conectan bloques es: BloqueDeOrigen\_BloqueDeDestino, Las señales que son de mas de un bit, incluyen su tamaño, que se muestra después de una línea diagonal así: /bits de la señal.

La descripción del funcionamiento de los bloques y las señales presentadas a continuación se realiza en la siguiente sección.

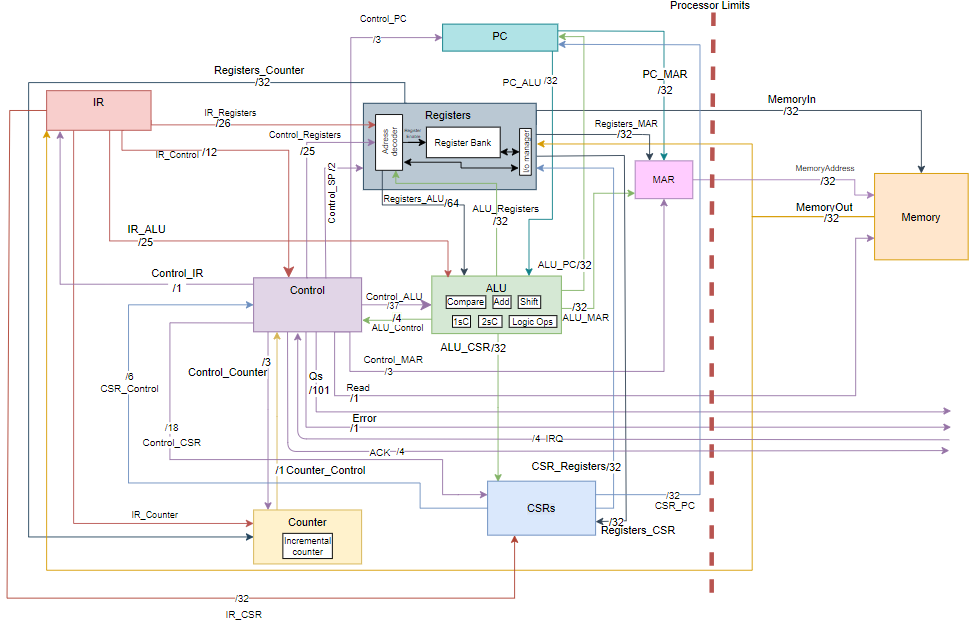


Figura. 5 Diagrama de bloques del procesador

## Descripción de bloques y señales

A continuación, se muestra la descripción de los bloques vistos en la Figura 5, así como la descripción de las señales que los comunican con los demás bloques:

**ProcessorRV**:

Bloque del procesador, contiene las entradas y salidas del sistema. En el diagrama para facilitar la notación, se muestra como la línea punteada que define los limites del procesador.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + MemoryOut: 32 bits. Datos provenientes de la memoria
  + IRQ: 4 bits. Bits de interrupciones para periféricos.
* Salidas
  + MemoryAddress: 32 Bits. señal que envía la dirección almacenada en MAR a la memoria.
  + Relojs: 1 Bit. Salida de la señal de reloj para análisis
  + error: 1 Bit. Señal de error en caso de no estar en ningún estado.
  + Qs: 102 Bits. Q de los estados para comprobación.
  + ACK:4 bits, bits de respuesta de interrupciones

**IR:**

Instruction Register, registro especial encargado de almacenar las instrucciones provenientes de la memoria y de enviar diferentes partes de la instrucción a los demás bloques para llevar a cabo las acciones necesarias.

* Entradas:
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + MemoryOut: 32 bits. Salida de datos de la memoria, en este caso, contiene la instrucción
  + Control\_IR: 1 bit. Señal del control que indica al IR reemplazar su valor con el valor de Memoryout
* Salidas:
  + IR\_Registers: 26 bits envía la dirección de los registros rd, rs1 o rs2 de las instrucciones para acceder a los registros necesarios. La cantidad de registros usados en la instrucción depende de cada instrucción, adicionalmente enviar datos inmediatos para almacenar.
    - Direccion rd: bits 0 : 5
    - Dato immediato bits 5 :25
    - Direccion rs1 bits 9 :13
    - Direccion rs2 bits 14:18
  + IR\_Control: 13 bits. Envían al control los opcodes de la instrucción para poder determinar los saltos de las instrucciones.
    - IR[25]|IR[30]|IR[15:12]|IR[6:0]
  + IR\_ALU: 25 bits. envía los datos que deben ser operados en la ALU, tales como datos inmediatos en las instrucciones.
    - IR\_ALU(24:0): IR (31 : 7)
  + IR\_CSRs: 32 bits. Envia las direcciones de los CSR donde se realiza la instrucción. O el IR completo para almacenarlo en las excepciones.
  + IR\_Counter : 6 bits. Se usa para cargar directamente un valor al Counter
    - IR\_Counter(5:0): (25:20)

**PC**:

Program Counter, registro espacial encargado de almacenar y aumentar la dirección de la memoria donde se recuperan las instrucciones, permite la carga de datos para hacer saltos, al iniciar empieza en la dirección 0x0000.

Los incrementos se dan de a 4.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + Control\_PC: 3 bits.
    - Control\_PC[0]: Señal de control que indica al PC que se auto incremente, estos auto incrementos aumentan 4 al PC actual.
    - Control\_PC[1]: Señal de control que indica al PC que reemplace su valor actual, con el valor de ALU\_PC.
    - Control\_PC[2]: Señal de control que indica al PC que reemplace su valor actual, con el valor de CSR.
  + ALU\_PC: 32 bits, contiene el nuevo PC para ser almacenado.
  + CSR\_PC: 32 bits, contiene el nuevo valor para PC.
* Salidas
  + PC\_MAR: 32 bits. Señal que envía la dirección desde el PC al MAR para obtener la instrucción de memoria.
  + PC\_ALU: 32 bits. Señal que envía el contenido del PC a la ALU.

**Registers**:

Bloque de registros, contiene el selector y 32 registros organizados de la siguiente manera, :

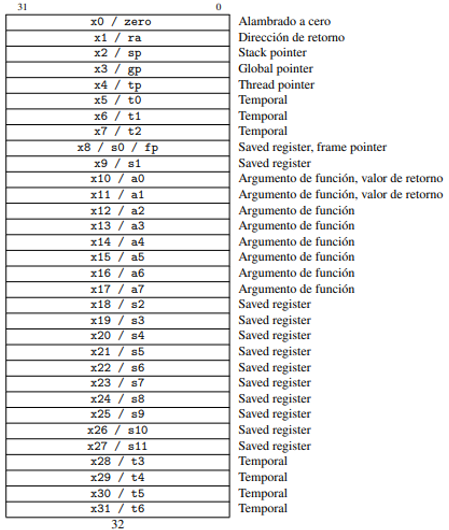


Figura. 6 Distribucion de registros

El bloque registers, posee un selector, que realiza el manejo organiza cuales de los registros son requeridos para esta instrucción, tanto para ser leídos o para almacenar datos dentro de ellos.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + IR\_Registers: 26 bits. Dirección de los registros a usar en la instrucción y en algunos casos datos immediate a guardar.
    - Bits 5:0 = IR(11:7)
    - Bits 10:6 = IR(19:15)
    - Bits 15:11 = IR(24:20)
    - Bits 20:16 = IR(6 : 2)
  + Control\_Registers: 25 Bits señal de control a los registros para indicar que datos se asignan a la salida y que datos se guardan así como la ubicación de estos, se detalla en los decoders.
  + ALU\_Registers: 32 bits. Señal de salida de la ALU, envía a los registros el resultado de una operación.
  + MemoryOut: 32 bits. Señal que lleva los datos de memoria a ser almacenados en los registros
  + CSR\_Registers: 32 bits Señal de los CSR a los registros que contiene el valor a guardar
* Salidas
  + Registers\_ALU: 64 bits. Señal que transmite los datos de los registros a la ALU para ser operados.
    - Registers\_ALU(63:32) segundo dato 32 bits
    - Registers\_ALU(31:0) primer dato 32 bits
  + Registers\_Counter: 32 bits. Señal que contiene la cantidad a contar.
  + Memory\_In: 32 bits, Señal que contiene el valor a guardar en memoria.
  + Registers\_CSR: 32 bits. Señal que contiene un dato a guardar en los CSR.
  + Registers\_MAR: 32 bits, señal del banco de registros al MAR para una nueva dirección.

El bloque de registros adentro posee:

* Decoder de instrucciones, que decodifica la dirección desde el IR para guardar los datos o desde que registros sacar los datos.
* Un banco de registros donde se almacenan los datos.
* SP un registro especial con dirección 2, más detalle en bloque SP.
* I/o manager asigna los datos de salida y de guardado según los estados del control.

**MAR:**

*Memory address register*. Registro de direcciones para almacenar o leer datos de la memoria.

* Entradas:
  + Reset: 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + Control\_MAR: 2 bits. Señal que maneja el comportamiento del MAR de acuerdo con los estados:
    - Bit 0: bit para que reciba Pc
    - Bit 1: bit para que reciba ALU
  + ALU\_MAR: 32 bits. Señal que envía el resultado de una operación de la ALU al MAR
  + PC\_MAR: 32 bits. Señal que envía la dirección desde el PC al MAR para obtener la instrucción de memoria.
  + Registers\_MAR: 32 bits, señal del banco de registros al MAR para una nueva dirección.
* Salidas:
  + MemoryAddress: 32 bits. Señal que envía la dirección almacenada en MAR a la memoria.

**Control**:

Bloque que contiene la máquina de estados del procesador, decide basado en las instrucciones y envía señales a los demás bloques para llevarlas a cabo.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + IR\_Control: 13 bits. Envía al control los opcodes de la instrucción para poder determinar los saltos de las instrucciones.
    - IR[25]|IR[30]|IR[15:12]|IR[6:0]
  + ALU\_Control: 4 bits
    - ALU\_Control[0] Bandera de fin de operación de la ALU
    - ALU\_Control[1] bandera de cero, se pone en uno si el resultado de una operación de ALU es cero
    - ALU\_Control[2] bit de signo resultado de ALU
    - ALU\_Control[3] bit de carry resultado de ALU
  + Counter\_Control:1 bit Señal que notifica el fin del conteo.
  + IRQ: 4 bits. Bits de interrupciones para periféricos.
  + CSR\_Control: 6 bits
* Salidas
  + Control\_MAR: 3 bits. Señal que maneja el comportamiento del MAR de acuerdo con los estados:
    - Bit 0: bit para que reciba Pc
    - Bit 1: bit para que reciba ALU
    - Bit 2: bit para que reciba registers
  + Control\_PC: 3 bits.
    - Control\_PC[0]: Señal de control que indica al PC que se auto incremente, estos auto incrementos aumentan 4 al PC actual.
    - Control\_PC[1]: Señal de control que indica al PC que reemplace su valor alctual, con el valor de ALU\_PC.
    - Control\_PC[2]: Señal de control que indica al PC que reemplace su valor alctual, con el valor CSR\_PC.
  + Control\_Registers: 25 Bits señal de control a los registros para indicar que datos se asignan a la salida y que datos se almacenan, así como la ubicación de estos, se detalla en los decoders.
  + Control\_IR: una señal de control para que reemplace su valor por el valor del cable Memoryout
  + Control\_ALU: 37 bits. Señal del control a la ALU, que dependiendo del estado indica cual operación, con que datos se lleva acabo y en donde se almacena. Para mas detalle se ve en el decoder.
  + Control\_Counter: 3 bits. Señal de control para el contador.
    - Control\_Counter[0]: Señal de control hacia el contador para iniciar la cuenta.
    - Control\_Counter[1]: Señal de control hacia el contador para reemplazar el valor de la cuenta por IR.
    - Control\_Counter[2]: Señal de control hacia el contador para reemplazar el valor de la cuenta por Registers.
  + Control\_SP: 2 bits señal para el comportamiento del SP.
    - Bit 0: Aumenta el conteo por 4.
    - Bit 1: Disminuye el conteo por cuatro.
  + Qs: 85 bits. Señal de los Q de la máquina de estados del control para verificar
  + Read: Señal para la memoria de escritura o lectura
    - Read = 1 se activa la lectura
    - Read = 0 se activa la escritura
  + Error: 1 bit Señal de error de la máquina de estados
  + Control\_CSR: 18 bits. Señal de activación de los CSR,
  + ACK:4 bits, bits de respuesta de interrupciones

**ALU**:

Aritmetic logic unit. Bloque que realiza las operaciones aritmeticas (suma, multiplicacion, corrimiento aritmetico, etc) y logicas (AND, OR XOR, etc ) entre datos ingresados y genera un resultado a la salida, la operación a llevar a cabo es determinada por una señal del control.

* Entradas:
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + Registers\_ALU: 64 bits. Señal que transmite los datos de los registros a la ALU para ser operados.
  + IR\_ALU: 25 bits. Envia los datos que deben ser operados en la ALU, tales como datos inmediatos en las instrucciones.
    - IR\_ALU(24:0): IR (31 : 7)
  + Control\_ALU: 37 bits. Señal del control a la ALU, que dependiendo del estado indica cual operación, con que datos se lleva acabo y en donde se almacena.
* Salidas:
  + ALU\_Registers: 32 bits. Señal de salida de la ALU, envia a los registros el resultado de una operación.
  + ALU\_PC: 32 bits, contiene el nuevo PC para ser almacenado.
  + ALU\_CSR: 32 bits. Señal de salida de la ALU, envia a los CSR el resultado de una operación.
  + ALU\_MAR: 32 bits señal que envía el resultado de una operación de la ALU al MAR
  + ALU\_Control: 4 bits
    - ALU\_Control[0] Bandera de fin de operación de la ALU
    - ALU\_Control[1] bandera de cero, se pone en uno si el resultado de una operación de ALU es cero
    - ALU\_Control[2] bit de signo resultado de ALU
    - ALU\_Control[3] bit de carry resultado de ALU

**CSRs**:

Control and Status Registers, bloque de registros útiles al sistema para desarrollar las instrucciones.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + ALU\_CSR: 32 bits. Señal de salida de la ALU, envia a los CSR el resultado de una operación.
  + IR\_CSRs: 12 bits. Envia las direcciones de los CSR donde se realiza la instrucción.
    - IR\_CSRs(11:0) : IR(31:20)
  + Registers\_CSR : 32 bits. Envia los datos para escribir en los CSR
  + Control\_CSR: 18 bits. Señal de activacion de los CSR
* Salidas
  + CSR\_PC: 32 bits, contiene el nuevo valor para PC.
  + CSR\_Registers : 32 bits. Envia a los registros datos
  + CSR\_Control: 6 bits

**Counter**:

Bloque de conteo, almacena un número y cuenta esa cantidad de veces, al terminal envía una señal a control.

* Entradas
  + Reset 1 Bit. Señal para reiniciar el sistema
  + Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
  + Registers\_Counter: 32 bits. Señal que contiene la cantidad a contar.
  + IR\_Counter : 6 bits. Se usa para cargar directamente un valor al Counter
    - IR\_Counter(5:0): (25:20)
  + Control\_Counter: 3 bits. Señal de control para el contador.
    - Control\_Counter[0]: Señal de control hacia el contador para iniciar la cuenta.
    - Control\_Counter[1]: Señal de control hacia el contador para reemplazar el valor de la cuenta por IR.
    - Control\_Counter[2]: Señal de control hacia el contador para reemplazar el valor de la cuenta por Registers.
* Salidas
  + Counter\_Control: 1 Bit. Señal que notifica el fin del conteo.

**SP:** Stack Pointer, registro que almacena la dirección de memoria donde se almacenan los datos para hacer llamados a subrutinas.

Entradas:

* + Control\_SP: 2 bits señal para el comportamiento del SP.
    - Bit 0: Aumenta el conteo por 4.
    - Bit 1: Disminuye el conteo por cuatro.

Salidas:

* + SPRegister: salida de 32 bits que contiene el valor actual de SP

Debido a que el stack pointer es un registro especial pero aun así esta implementado en el bloque de registros, para escribir la información de este o usarlo para operaciones se usan las operaciones sobre el bloque de registros con la dirección 0 T 010.

## Descripcion en RTL

Se realiza una descripción en lenguaje de transferencia de registros en base a las instrucciones seleccionadas, se descompone cada instrucción en una serie de pasos individuales, que se realizan en sucesión para lograr realizar la instrucción en específico.

Cada flujo en el procesador comienza por obtener una instrucción de la memoria, en una etapa de fetch, seguido por la decodificación de esta instrucción en el control, que de acuerdo al código de operación encuentra cual es la instrucción que se esta ejecutando, saltando a los estados que ejecutan los pasos necesarios, estos estados envían señales del control a los diferentes bloques del sistema para activarlos cuando sea necesario usarlos para ejecutar la instrucción. En la decodificación del control se realiza también la verificación de la instrucción, en caso de que la instrucción no sea válida, se ejecuta una serie de pasos diseñados para informar al usuario de la instrucción no valida y crear los registros correspondientes en el bloque CSR, tras esto se continua con el flujo normal.

Al final del flujo de cada instrucción la maquina de estados verifica que no haya interrupciones y en caso de haberlas si las interrupciones para el procesador se encuentran habilitadas, con los bits de interrupciones del registro correspondiente del CSR en 1, se atiende la interrupción. El procesador implementa interrupciones auto vectorizadas por lo que al recibir una interrupción se refiere a un segmento especifico de la memoria donde deben estar almacenadas las instrucciones a ejecutar por cada interrupción.

La descripción completa en pasos de cada una de las instrucciones, la forma en que se decodifican se estructura el AHPL agrupando instrucciones similares y la cantidad de ciclos necesarios para llevar a cabo las instrucciones se muestran a detalle en el AHPL documentado, incluido como anexo a este documento.

1. IMPLEMENTACIÓN

En esta sección, se detalla la fase de implementación del proyecto. Mientras la sección anterior,

aquí se explica

1. APRENDIMOS A USAR GITHUB
2. DIVISION DE BLOQUES
3. DISEÑO INDIVIDUAL DE BLOQUES
4. ESPECIFICACION ARQUITECTURAL DE BLOQUES SIGNIFICATIVOS
   1. BLOQUE DE SUMA
   2. BLOQUE DE MULTIPLICACION
   3. REGISTROS
   4. ALU
   5. CONTROL
5. ¿ESQUEMATICOS?
6. RESULTADOS

SIMULACIONES

RESULTADOS DE IMPLEMENTACIÓN FISICA

1. CONCLUSIONES

## Impacto y análisis del proyecto.

## Conclusions and Future Work

Implementar diferentes modos en el procesador y con ello todas las instrucciones base del estándar RV32I

*.*

*Al usar el estandar risc v, se ha comprobado su utilidad para el desarrollo de procesadores, resolviendo varios elementos del diseño de un procesador, como la creación de instrucciones, los modos de direccionamiento y , pero su uso para proyectos de corta duración debe ser estudiado cuidadosamente, la aplicación del estándar requiere de un cuidadoso y profundo estudio previo de este, además algunas caracteristicas están disponibles solo para algunas versiones del estándar, así que la selección del estándar a usar juega un papel importante de la arquitectura.*

*\*\*\*separar en dos conclusiones*

1. REFERENCIAS

[1] A. L. S. Iglesias, “¿Qué es la arquitectura de un procesador?”, *aboutespanol*. https://www.aboutespanol.com/que-es-la-arquitectura-de-un-procesador-841131 (consultado jun. 16, 2020).

[2] “MIPS Open – MIPS”. https://www.mips.com/mipsopen/ (consultado jun. 16, 2020).

[3] “RISC-V ISA”, *RISC-V Foundation*. https://riscv.org/risc-v-isa/ (consultado sep. 29, 2019).

[4] PixhawkAdmin, “Home Page”, *Pixhawk*. https://pixhawk.org/ (consultado sep. 21, 2019).

[5] I. Sizova, “GAP application processors”, *GreenWaves Technologies*. https://greenwaves-technologies.com/gap8\_gap9/ (consultado jun. 16, 2020).

[6] “Qualcomm Flight Pro”, *Qualcomm Developer Network*. https://developer.qualcomm.com/hardware/qualcomm-flight-pro (consultado sep. 21, 2019).

[7] “Why RISC-V?”, *RISC-V Foundation*. https://riscv.org/why-risc-v/ (consultado sep. 29, 2019).

[8] “RM0090 Reference manual”. Consultado: jun. 16, 2020. [En línea]. Disponible en: https://www.st.com/content/ccc/resource/technical/document/reference\_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf.

[9] “STM32 32-bit Arm Cortex MCUs”, *STMicroelectronics*. https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html (consultado sep. 21, 2019).

[10] W. Chen, Y. Dong, y Z. Duan, “Manipulating Drone Position Control”, en *2019 IEEE Conference on Communications and Network Security (CNS)*, jun. 2019, pp. 1–9, doi: 10.1109/CNS.2019.8802817.

[11] H. Chao, Y. Cao, y Y. Chen, “Autopilots for small unmanned aerial vehicles: A survey”, *Int. J. Control Autom. Syst.*, vol. 8, núm. 1, pp. 36–44, feb. 2010, doi: 10.1007/s12555-010-0105-z.

[12] “El controlador PID.pdf”. Consultado: jun. 16, 2020. [En línea]. Disponible en: http://www.dia.uned.es/~fmorilla/MaterialDidactico/El%20controlador%20PID.pdf.

[13] Oscar, “Quadcopter PID Explained”, *Oscar Liang*, feb. 08, 2018. https://oscarliang.com/quadcopter-pid-explained-tuning/ (consultado nov. 19, 2019).

[14] “Metodologias\_de\_diseno.pdf”. Consultado: ago. 31, 2019. [En línea]. Disponible en: https://moodle2.unid.edu.mx/dts\_cursos\_mdl/lic/IC/EA/AM/06/Metodologias\_de\_diseno.pdf.

[15] “▷ Qué es un procesador y cómo funciona”, *Profesional Review*, oct. 30, 2018. https://www.profesionalreview.com/2018/10/30/que-es-un-procesador/ (consultado jun. 17, 2020).

[16] “What is an Instruction Set?” https://www.computerhope.com/jargon/i/instset.htm (consultado jun. 17, 2020).

[17] C. A. E. II, *Concise Encyclopedia of System Safety: Definition of Terms and Concepts*. John Wiley & Sons, 2011.

[18] “Definition of closed architecture”, *PCMAG*. https://www.pcmag.com/encyclopedia/term/closed-architecture (consultado jun. 17, 2020).

[19] “Arquitectura Computacional.pdf”. Consultado: jun. 17, 2020. [En línea]. Disponible en: http://homepage.cem.itesm.mx/garcia.andres/PDF201411/Arquitectura%20Computacional.pdf.

[20] “Manual Intel Aero”. Consultado: jun. 17, 2020. [En línea]. Disponible en: https://docs.rs-online.com/2ab8/0900766b815d6e2a.pdf.

[21] “The history of Pixhawk • Auterion”, *Auterion*. https://auterion.com/the-history-of-pixhawk/ (consultado jun. 17, 2020).

[22] “RISC-V History”, *RISC-V International*. https://riscv.org/risc-v-history/ (consultado jun. 17, 2020).

[23] “Open Source Drones”. http://www.yolinux.com/TUTORIALS/Drones.html (consultado jun. 17, 2020).

[24] “GAP8 - GreenWaves - WikiChip”. https://en.wikichip.org/wiki/greenwaves/gap8 (consultado jun. 17, 2020).

[25] E. Flamand *et al.*, “GAP-8: A RISC-V SoC for AI at the Edge of the IoT”, en *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, jul. 2018, pp. 1–4, doi: 10.1109/ASAP.2018.8445101.

[26] “ArduPilot :: About”. https://ardupilot.org/index.php/about (consultado jun. 17, 2020).

[27] A. Ltd, “Cortex-M4”, *Arm Developer*. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4 (consultado jun. 17, 2020).

[28] “Atmel-7810-Automotive-Microcontrollers-ATmega328P\_Datasheet.pdf”. Consultado: jun. 17, 2020. [En línea]. Disponible en: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\_Datasheet.pdf.

[29] “guia-practica-de-risc-v-1.0.5.pdf”. Consultado: jun. 16, 2020. [En línea]. Disponible en: http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf.

1. APÉNDICES

NOTA: Lo que ponemos aquí son cosas utiles para entender el TG, pero recordemos que los jurados no van a mirar ningún apéndice, porque el documento debe ser autocontenido.