REPORTE DE PRÁCTICA NO.2

Diseño de un microprocesador de arquitectura Harvard

UNIVERSIDAD AUTÓNOMA DE ZACATECAS

UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN ROBÓTICA Y MECATRÓNICA

Microcontroladores y Lab.

Dr. Remberto Sandoval Aréchiga

Equipo:

-Francisco Javier Ruiz Bravo Hurtado

-Francisco Javier López Vásquez

-Ana Sofía Martínez Murillo

-José Rosario Guerrero Flores

Grupo: 5°A IRM **Entrega:** 24 de Noviembre del 2020

OBJETIVO

Diseñar un microprocesador con arquitectura Harvard y un set de instrucciones RISC haciendo uso de la metodología de diseño propuesta Top Down y de las herramientas de descripción de hardware en FPGA.

RESUMEN

En esta práctica se realizó un primer acercamiento al proceso de diseño de un hardware basado en una arquitectura Harvard, en la que la memoria de instrucciones y la memoria de datos se encuentran separadas y las operaciones se realizan directamente en los registros del microprocesador, ofreciendo ventajas como la paralelización de procesos, la optimización de los ciclos de fetching y executing del sistema, así como la liberación del bus de datos.

También se hizo uso de un set de instrucciones basado en una arquitectura de instrucciones reducidas o RISC con la finalidad facilitar la ejecución de tareas por parte del microprocesador, lo que ofrece muchas ventajas en el rendimiento energético de este.

El problema planteado fue el procesamiento de datos a partir de instrucciones reducidas brindadas por programas almacenados, realizando una o varias operaciones con el fin de cumplir con la tarea determinada por el programa.

La solución propuesta consiste en el entrelazamiento de distintos módulos con funciones específicas para el control de los componentes internos y de la secuencia de instrucciones, lectura, operación y almacenaje de datos, así como para la comunicación con las memorias tanto de datos como de instrucciones y la gestión de los datos de salida.

La metodología utilizada para el diseño del microprocesador fue la Top-Down.

ÍNDICE

INTRODUCCIÓN

Iniciamos con el análisis del problema, se requería hacer un microprocesador de 8 bits, y la arquitectura propuesta para realizarlo fue la arquitectura Harvard debido a que la separación de ambas memorias, la de instrucciones y la de datos permitía la paralelización de procesos, así como la liberación de ancho de bus, también se propuso un set compuesto de 8 instrucciones basado en la arquitectura RISC, la cual es utilizada actualmente en la mayoría de los microprocesadores móviles debido a su alto desempeño y bajo consumo energético, una de las ventajas del uso de la arquitectura RISC es que a pesar de que se basa en instrucciones reducidas lo que conlleva a un mayor número de ciclos de reloj para realizar una instrucción compleja, divide dicha instrucción en otras más sencillas, lo que disminuye la carga en el microprocesador, debido a que las posibilidades de resolución de una instrucción en un procesador RISC es mucho menor a la de uno con arquitectura CISC.

El microprocesador a diseñar debía realizar las operaciones de carga inmediata e indirecta de datos, almacenaje de datos en memoria, movimiento de los datos en los registros, las operaciones matemáticas de suma, resta, corrimientos, las operaciones lógicas NOT, AND y OR así como la potenciación de los datos, saltos condicionales e incondicionales y finalmente una instrucción sin operación.

Después de un análisis se establecieron las entradas del microprocesador, las cuales fueron, una entrada de la memoria de instrucciones denominada i_Instrucciones cuyo ancho de bus sería de 9 bits ya que cada instrucción se compone de 3 partes, el operation code y dos operandos; una entrada de datos denominada i_Bus_de_datos cuyo ancho sería de 8 bits (definido por los 8 bits del microprocesador), además de las entradas de sistema i_Clk e i_Rst. Una entrada más que se propuso con el fin de realizar pruebas en el microprocesador, a la cual se le nombró i_Frec_de_trabajo de 32 bits cuya función es establecer el límite de un preescalador interno. Las salidas del microprocesador serían o_Direciones_Instrucciones (Bus de direccionamiento a la memoria de instrucciones), o_Direcciones_Datos (Bus de direccionamiento de la memoria de datos), o_Bus_de_Datos (Bus de datos que ingresa a la memoria de datos) y por último una señal que marca si se realizará un proceso de escritura o de lectura en memoria denominado o_Lectura_Escritura.

MARCO TEÓRICO

Se necesitarán los siguientes conceptos:

-Microprocesador: El microprocesador es el circuito integrado central más complejo de un sistema informático. Es el encargado de ejecutar los programas, desde el sistema operativo hasta las aplicaciones de usuario; solo ejecuta instrucciones programadas en lenguaje de bajo nivel, realizando operaciones aritméticas y lógicas simples, tales como sumar, restar, multiplicar, dividir, las lógicas binarias y accesos a memoria.

Puede contener una o más unidades centrales de procesamiento (CPU) constituidas, esencialmente, por registros, una unidad de control, una unidad aritmético lógica (ALU).

-Arquitectura de microprocesadores: El microprocesador tiene una arquitectura parecida a la computadora digital. En otras palabras, el microprocesador es como la computadora digital porque ambos realizan cálculos bajo un programa de control. Consiguientemente, la historia de la computadora digital ayuda a entender el microprocesador. Hizo posible la fabricación de potentes calculadoras y de muchos otros productos. El microprocesador utiliza el mismo tipo de lógica que es usado en la unidad procesadora central (CPU) de una computadora digital.

-Arquitectura Harvard: La arquitectura de Harvard es una arquitectura de computadora con pistas de almacenamiento y de señal físicamente separadas para las instrucciones y para los datos. En la arquitectura Harvard, no hay necesidad de hacer que las dos memorias compartan características. En particular, pueden diferir la anchura de palabra, el momento, la tecnología de implementación y la estructura de dirección de memoria. En algunos sistemas, se pueden almacenar instrucciones en memoria de solo lectura mientras que, en general, la memoria de datos requiere memoria de lectura-escritura. En algunos sistemas, hay mucha más memoria de instrucciones que memoria de datos así que las direcciones de instrucción son más anchas que las direcciones de datos.

En la actualidad la mayoría de los procesadores implementan dichas vías de señales separadas por motivos de rendimiento, pero en realidad implementan una arquitectura Harvard modificada, para que puedan soportar tareas tales como la carga de un programa desde una unidad de disco como datos para su posterior ejecución.

Set de instrucciones: La arquitectura de un microprocesador o de cualquier procesador se define por el conjunto de instrucciones que puede obedecer, las maneras en que las instrucciones pueden especificar la localización de los datos por procesar. Las instrucciones que obedece un microprocesador están codificadas como dígitos binarios en un sistema de memoria, cada instrucción se divide en uno o más campos, todas las instrucciones tienen un campo de código de operación que define el propósito de instrucción como sumar o mover datos.

Set de instrucciones RISC: Es un tipo de diseño de CPU generalmente utilizado en microprocesadores o microcontroladores con instrucciones de tamaño fijo y presentadas en un reducido número de formatos además de que solo las instrucciones de carga y almacenamiento acceden a la memoria de datos. Estos procesadores suelen disponer de muchos registros de propósito general.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC,2 DEC Alpha, MIPS, ARM, SPARC son ejemplos de algunos de ellos.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse.

Ancho de banda del bus de datos: En su forma más simple, el ancho de banda es la capacidad de transferencia de datos, en otras palabras, la cantidad de datos que se pueden mover de un punto a otro en cierta cantidad de tiempo. Este dato determina el número de bits que es capaz de procesar el microprocesador.

Registros: Un registro es una memoria de alta velocidad y poca capacidad, integrada en el microprocesador, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas.

Los registros están en la cumbre de la jerarquía de memoria, y son la manera más rápida que tiene el sistema de almacenar datos. Los registros se miden generalmente por el número de bits que almacenan, generalmente se implementan en un banco de registros, pero antiguamente se usaban biestables individuales, memoria SRAM o formas aún más primitivas.

El término es usado generalmente para referirse al grupo de registros que pueden ser directamente indexados como operandos de una instrucción, como está definido en el conjunto de instrucciones. Sin embargo, los microprocesadores tienen además muchos otros registros que se usan con un propósito específico, como el contador de programa.

DESARROLLO

Debido que la arquitectura Harvard se caracteriza por la manipulación de los datos en los registros se propuso un esquema de solución basado en un banco de registros cuyas funciones se dividirían según el tipo de dato a manipular, una instrucción o un dato, por lo que se crearon los componentes Registro_de_instrucciones y Registro_de_Datos cuya función se pueden inferir por su nombre, el registro de instrucciones se encarga de almacenar la instrucción a ejecutar y dividirla en 2 partes, el operation code y los operandos, también permite trabajar con las instrucciones de carga y almacenaje inmediato de datos, por otro lado, los registros de datos se encargan de almacenar los datos en cualquiera de sus 8 registros, que son el número total de registros que se pueden direccionar con uno solo de los operandos, además permite los procesos de escritura de registros, lectura de registros, movimiento de registros y de lectura y escritura simultánea con el fin de poder realizar todas las posibles operaciones dentro del microprocesador. Ambos componentes descritos previamente al ser bancos de registros, se diseñaron como componentes secuenciales, por que están sujetos a la señal de reloj del dispositivo.

El segundo componente desarrollado fue un contador de sistema al cual llamamos PC, este componente se trata de un registro auto incrementable con cada ciclo de reloj, el cual almacena la instrucción a ejecutar y se comunica directamente con la salida del o_Direcciones_Instrucciones para realizar el proceso de direccionamiento de la instrucción, este componente también permite realizar saltos de instrucciones por lo que cuenta con las entradas i_Direccion_salto e i_Señal_de_salto las cuales señalizan la ejecución de un salto y brindan la dirección de la instrucción a saltar respectivamente, pero no sin antes mandar a almacenar la instrucción en la que el contador se quedó, la cual se dirige al banco de registros a través de la salida o_Señal_a_stack.

Con el fin de poder realizar los procesos de lectura y escritura en memoria, así como de direccionamiento de datos, se diseñó un componente combinacional basado en la anidación de multiplexores, a este se le llamó Manager_de_salidas_a_memoria y cuyo trabajo es

seleccionar dependiendo de una señal de control i_Señal_de_control, las señales internas que pasarán sus respectivas salidas a memoria, dependiendo si es un proceso de lectura o escritura y si se tiene una instrucción con direccionamiento inmediato.

Obtenida del análisis de otras arquitecturas de microprocesadores, se diseñó una unidad lógico aritmética combinacional denominada ALU, la cual se encarga de realizar todas las operaciones matemáticas y lógicas del microprocesador durante la instrucción MATH, así como la asignación de banderas dependiendo de los resultados de la operación realizada, esto con el fin de que otro componente posteriormente habilite la función de saltos condicionales dependiendo del estado de dichas banderas. Los resultados de las operaciones realizadas en esta unidad, son almacenadas en los registros de datos.

Con el fin de permitir el control y gestión de todos los procesos dentro de los componentes del microprocesador así como la sincronización entre ellos, se desarrolló una estructura de control denominada Unidad_de_Control, la cual como ya se mencionó se encarga de verificar la correcta ejecución de las operaciones dentro del microprocesador con base en las instrucciones recibidas, esta unidad está estrechamente ligada al componente de Registro_de_Datos debido a que realiza una decodificación de señales que le permite realizar todos los procesos de gestión de registros. Esta unidad es el cerebro del microprocesador, se encarga de la evaluación del estado de todos los componentes internos, permite la ejecución de saltos condicionales e incondicionales en el PC a partir de las banderas otorgadas por la ALU, el control de las salidas a memoria del Manager_de_salidas_a_memoria y finalmente la selección de los datos que ingresan a los registros de datos.

Como ya se mencionó anteriormente, la Unidad de control gestiona los datos que ingresan a los registros de datos, para ello se ayuda de un Multiplexor de 4 a 1 el cual selecciona entre las señales provenientes de componentes como la ALU, el PC y el registro de instrucciones, así como los datos provenientes del bus de datos de entrada, para disminuir el número de entradas a los registros con el fin de aligerar su carga de trabajo y disminuir la complejidad del componente.

El último componente diseñado fue un preescalador, cuyo límite es establecido por la entrada i_Frec_de_trabajo, su finalidad es la de ser un componente que permita modificar la frecuencia a la que trabajan los componentes internos del microprocesador con la finalidad de realizar pruebas y análisis.

El funcionamiento del microprocesador es el siguiente:

-Al establecer la señal de reset i_Rst en alto, el componente PC iniciará el direccionamiento de la primera instrucción a través de la salida o_Direcciones_Instrucciones.

-La instrucción solicitada en el proceso de fetch, ingresará por la entrada i_Instrucciones al microprocesador y posteriormente al Registro_de_instrucciones donde se almacenará temporalmente y se separará en su Operation code (Los 3 bits más significativos) y en operandos (Los 6 bits restantes). En caso de que la instrucción sea una carga inmediata en registros o un almacenamiento inmediato en memoria, asignará los 3 bits menos significativos a la salida o_Direccionamiento_inmediato, la cual se conecta al Manager_de_salidas_a_memoria (en caso de un almacenamiento inmediato) y al multiplexor de entrada a los registros de datos (Para una carga inmediata de registros). Las salidas o_Operandos y O_Operation_code se conectan a la Unidad_de_control.

-La unidad de control a partir de las entradas i_Operation interpreta la instrucción a realizar, con la entrada i_Operandos realiza el direccionamiento para la gestión de los registros de datos tomando los primeros 3 bits para el primer operando y los restantes para el segudo, en caso de una operación MATH toma los 3 bits menos significativos de la entrada i_Operandos y los conecta a la ALU a partir de la señal Inst_decodificada. En caso de una instrucción JUMP, toma dichos 3 bits menos significativos para evaluar la condición de salto junto con la entrada i_Banderas, para que, a partir de la salida o_Señal de salto, realice un salto condicional a través de l PC. Todas las salidas a memoria son controladas a partir de la salida o_Señal_de_control, conectada al Manager_de_slaidas_a_memoria.

-El mutiplexor de entrada a los registros de datos MUX, permite la selección de la cadena de datos que ingresará a dicho componente, esto lo realiza a partir de la señal de

Selector_de_entrada_a_registros proveniente de la unidad de control. Este componente debe multiplexar las señales de Resultado (Proveniente de la ALU para almacenar el resultado de una operación en el registro R0), Direccionamiento_Inmediato (Proveniente del registro de instrucciones en caso de una carga inmediata de datos en los registros), i_Bus_de_datos (El bus de datos de entrada desde la memoria de datos) y Señal_a_stack (Proveniente del PC y que contiene la dirección de la instrucción en la que se quedó el PC antes de realizar un salto condicional o no).

-Los Registros_de_datos reciben la entrada multiplexada por el MUX y a partir de las señales de control y gestión de registros provenientes de la unidad de control realiza las operaciones correspondientes al movimiento, lectura o escritura de registros. La entrada i_Lectura_escritura establece si se llevará a cabo un proceso de lectura (“00”), un proceso de escritura (“01”), un movimiento de registros (“10”) o un proceso de lectura y escritura simultáneos (“11”). En caso de un proceso de lectura, la entrada i_Control_RX direcciona el registro que contiene el valor que se pasará a la salida o_RX, la entrada i_Control_RY direcciona el registro que contiene el valor que se pasará a la salida o_RY. En caso de un proceso de escritura, la entrada i_Seleccion_de_resitro_de_escritura direccionará el registro en el que se almacenarán los datos de la entrada i_Datos. En caso de un movimiento de registros, la entrada i_Seleccion_registro_lectura, direccionará el registro del cual se obtendrán los datos a mover y la entrada i_Seleccion_de_resitro_de_escritura direcciona el registro en el que se almacenará dicho dato. Y por último, si se tiene una operación de lectura y escritura simultánea de registros (La cual se da en caso de las instrucciones JUMP, LOAD2, STORE1, STORE2 y MATH) la entrada i_Control_RX direccionará el registro cuyo valor se pasa a la salida o_RX, lo mismo pasa con la entrada i_Control_RY pero para la salida o_RY y la entrada i_Seleccion_de_resitro_de_escritura direcciona el registro en el que se escribirán los datos provenientes de la entrada i_Datos. Las señales RX y RY se conectan al Manager_de_salidas_a_memoria para direccionamiento y almacenamiento de datos en memoria y a la ALU, para la operación de datos.

-La ALU es habilitada por la señal i_Hab proveniente de la unidad de control, la entrada i_Inst_decodificada contiene el número de la operación lógica o matemática que realizará

y las entradas i_RX e i_RY contienen los datos a operar. Una vez operados los datos, el resultado se pasa a la salida o_Resultado para su almacenamiento en registros, no sin antes ser analizado para, asignar las banderas de acarreo, resultado negativo o resultado cero, las cuales se concatenan y se pasan a la salida o_Banderas la cual se conecta a la unidad de control.

-El Manager_de_salidas_a_memoria basa sus operaciones en la entrada i_Señal_de_control proveniente de la unidad de control, esta entrada posee 3 bits, el primer bit visto desde el más significativo, habilita el funcionamiento de este componente, en caso de estar en alto, el componente tomará en cuenta las demás entradas, el segundo bit determina si se realizará un proceso de lectura o escritura en memoria (‘0’ escritura, ‘1’ lectura) y el bit menos significativo determina si se realizará un direccionamiento inmediato o indirecto en memoria (‘1’ Inmediato, ‘0’ indeirecto). Proceso de lectura, el contenido de la entrada i_RY direccionará el dato a solicitar en memoria en la salida o_Direcciones_Datos y la salida o_Lectura_Escritura será ‘0’, en caso de un proceso de escritura por direccionamiento indirecto, la entrada i_RX determinará la dirección de la posición de memoria en donde se desea escribir, la entrada i_RY contendrá el dato a escribir a través de la salida o_Bus_Datos y la salida o_Lectura_Escritura será ‘1’, por último, en caso de un proceso de escritura por direccionamiento inmediato, la entrada i_Direccionamiento_inmediato contendrá el dato a escribir, mientras i_RX contendrá la dirección, igualmente o_Lectura_Escritura será ‘1’.

ARQUITECTURA

Diagrama de caja negra

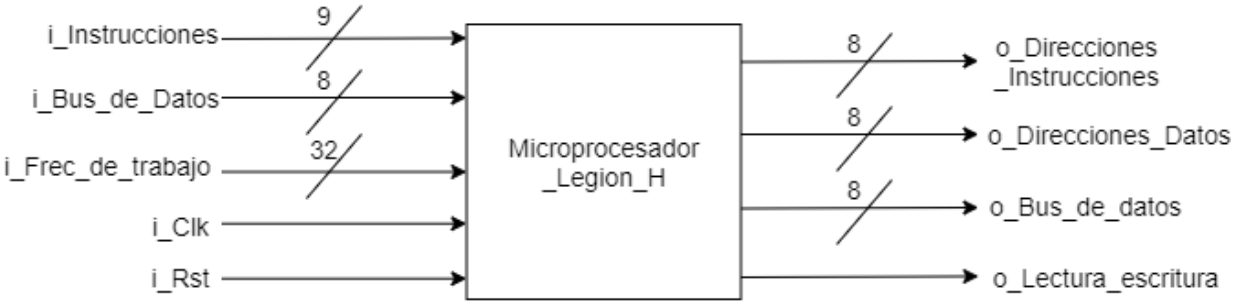

Diagrama 1: Diagrama de caja negra/ Top del sistema

Entradas y salidas del microprocesador

Señal	Núm. de bits	Descripción
i_Instrucciones	8 bits	Bus de instrucciones de entrada, señal de entrada que se encarga de traer los datos de las instrucciones desde la memoria ROM hasta el microprocesador.
i_Bus_de_Datos	8 bits	El bus de datos de entrada es una señal de entrada que se encarga de traer los datos desde la memoria de datos al microprocesador.
i_Frec_de _trabajo	32 bits	Entrada que referencia a la frecuencia de trabajo, debe ser un valor almacenado en memoria.
i_Clk	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia de 100 MHz.
i_Rst	8 bits	Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.

o_Direcciones _Instrucciones	8 bits	El bus de direcciones de instrucciones es el bus de direcciones que lleva las solicitudes de instrucciones desde el microprocesador a la memoria de instrucciones o ROM
o_Direcciones _de_Datos	8 bits	El bus de direcciones de datos se encarga de llevar las direcciones de los datos que el microprocesador requiere a la memoria de datos.
o_Bus_de_Datos	8 bits	El bus de datos de salida se encarga de llevar los datos que el microprocesador desea escribir en la memoria de datos.
o_Lectura _escritura	1 bit	Le señala a la memoria de datos si se va a leer un dato o se va a escribir sobre ella.

Tabla 1: Entradas del sistema Top del microprocesador

El diagrama 1 esquematiza el primer diseño del microprocesador, el cual es un diseño de caja negra, propio de la metodología Top Down en el cual únicamente se representan las entradas existentes y las salidas requeridas, pero sin plantear aún los componentes internos para la manipulación de los datos.

Las entradas al microprocesador son las siguientes: Una entrada de la memoria de instrucciones denominada i_Instrucciones de 9 bits; una entrada de datos denominada i_Bus_de_datos cuyo ancho es de 8 bits), además de las entradas de sistema i_Clk e i_Rst e i_Frec_de_trabajo de 32 bits cuya función es establecer el límite de un preescalador interno. Las salidas del microprocesador serían o_Direciones_Instrucciones (Bus de direccionamiento a la memoria de instrucciones), o_Direcciones_Datos (Bus de direccionamiento de la memoria de datos), o_Bus_de_Datos (Bus de datos que ingresa a la memoria de datos) y por último una señal que marca si se realizará un proceso de escritura o de lectura en memoria denominado o_Lectura_Escritura.

Set de instrucciones

Instrucciones	Argumentos	Descripción	Comentarios
LOAD 1	RX, #NUM	Carga #NUM en el registro RX.	#NUM es de 3 bits y se almacena en registros de 8 bits [0,7].
LOAD 2	RX, [RY]	Carga el dato de la dirección de memoria [RY] en el registro RX.	RX y RY son de 3 bits referenciando registros de 8 bits [0,7].
STORE 1	RX, #NUM	Almacena #NUM en la dirección de memoria [RX].	#NUM es de 3 bits y se almacena en memoria como un dato de 8 bits [0,7].
STORE 2	[RX], RY	Almacena el dato del registro RY en la dirección [RX] de memoria.	RY y RX son de 3 bits y hacen referencia a registros de 8 bits [0,7].
MOVE	RX, RY	Mueve los datos del registro RY al registro RX.	RY y RX son de 3 bits y hacen referencia a registros de 8 bits [0,7].
MATH	RX, OP	Realiza una operación matemática OP con los datos	OP: 0: R0=R0+RX 1: R0=R0-RX

		del registro RX y almacena en el registro R0.	2: R0=R0<<RX 3: R0=R0>>RY 4: R0=~RX 5: R0=R0&RX 6: R0=R0|RX 7: R0=R0^RX
JUMP	[RX], COND	Realiza un salto a la dirección	COND: 0: Sin condición 1: Sin condición, guarda el PC en el registro R7. 2: Bandera Z verdadera 3: Bandera Z falsa 4: Bandera C verdadera 5: Bandera C falsa 6: Bandera N verdadera 7: Bandera N falsa
NOP	-	Sin operación	-

Tabla 2: Set de instrucciones utilizado

Diagrama de caja blanca
V2.0

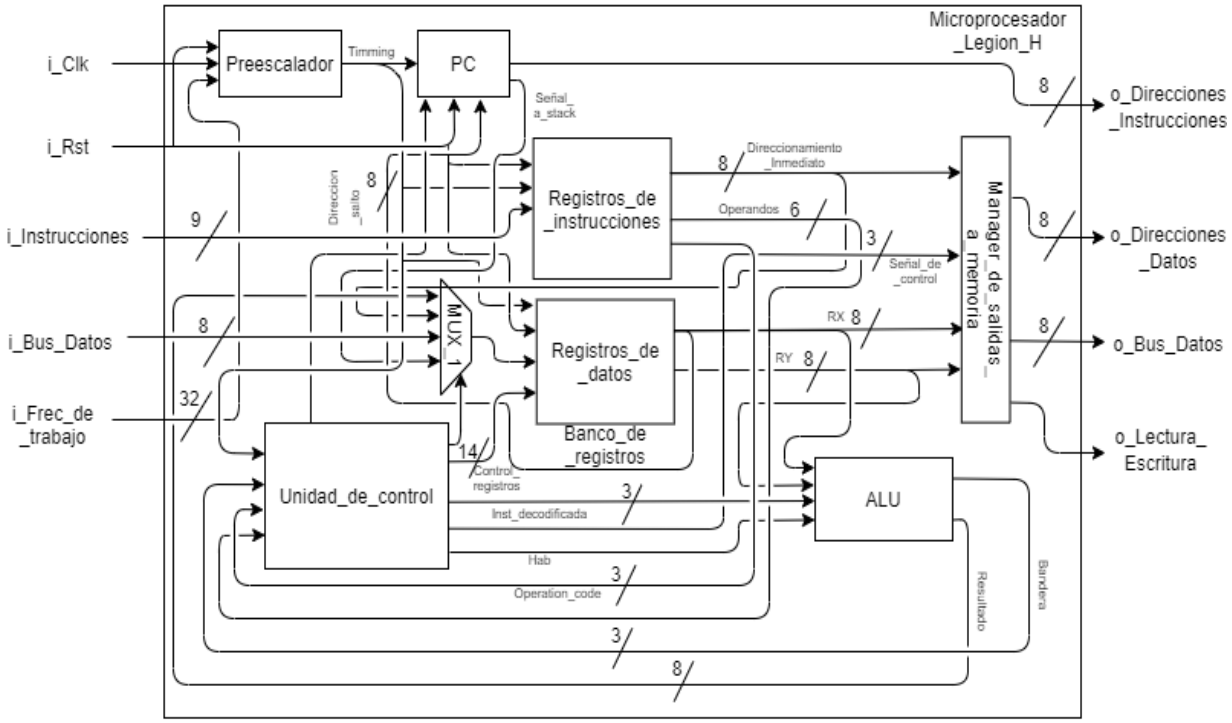

Diagrama 2: Diagrama de caja blanca

Componentes internos

Bloque	Función
Preescalador	Se encarga de dividir la frecuencia de entrada para adaptarla a la frecuencia de funcionamiento de los componentes internos, otorga una señal lógica cuyo fin será sincronizar los procesos del sistema.
PC	El PC counter que funge como puntero de instrucciones, se encarga de registrar e indicar la posición del procesador en la secuencia de instrucciones e incrementa con cada ciclo de la señal timming, en caso de romper con la secuencia se ayuda con un registro de stack o

	pila para guardar la posición en la que se encontraba para posteriormente volver a ella.
Registros_de _instrucciones	Decodifica las instrucciones que llegan desde el registro de instrucciones y actúa conforme al set de instrucciones establecido y coordina las operaciones en la unidad lógica aritmética, también analiza el estado de las banderas y conforme a este establece un salto o no en el PC counter o envía una señal para que se pase algún valor almacenado en el registro al bus de datos para su posterior almacenamiento en memoria
Registros_de _datos	Permite el almacenamiento de los datos provenientes de la memoria de datos y el procesamiento de estos, guarda los resultados de las operaciones realizadas por la ALU y guarda la dirección previa al salto del PC.
ALU	Se encarga de realizar las operaciones matemáticas y lógicas para enviar los datos de las operaciones a la unidad de control y posteriormente poder almacenar los resultados en los registros. También se encarga de establecer banderas o avisos que le servirán a la unidad de control para evaluar el estado de la operación, las banderas que contiene pueden ser de acarreo, de cero, de desbordamiento y de signo.
Manager_de _salidas _a_memoria	Se encarga de manejar el direccionamiento directo, indirecto o inmediato en el bus de direcciones de salida o_Direcciones_Datos, se guía de la señal de control para pasar datos desde una salida de registros al bus de datos de salida o_Bus_Datos o establecer un estado de alta impedancia.

MUX_1	Se encarga de decidir a partir de una señal de control qué datos entran a los registros de datos, si no hay ningún resultado de operación por parte de la ALU, permite la entrada de datos desde la memoria de datos, en caso contrario, pasa el resultado de la operación, de la señal a stack del PC counter o

Tabla 3: Componentes internos del microprocesador

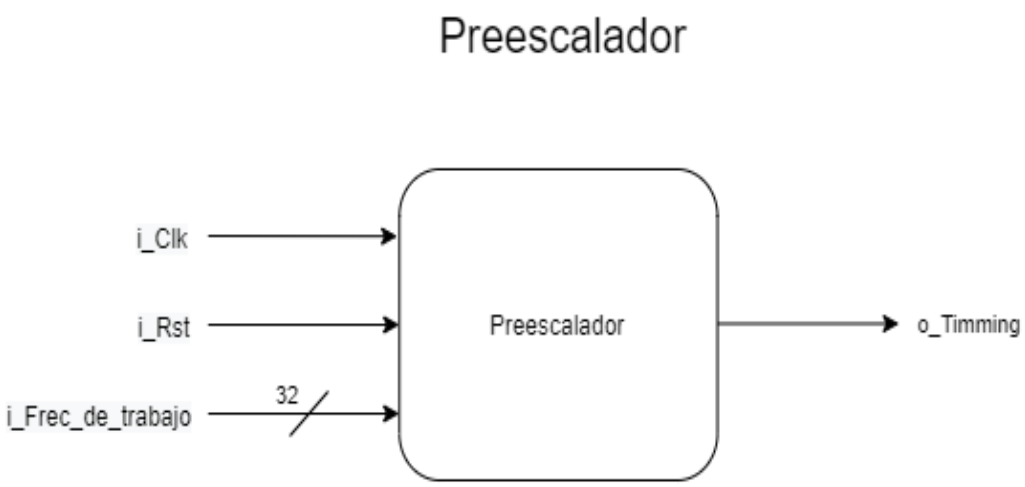

Diagrama 3: Preescalador

Entradas y salidas

Señal	Núm. de bits	Descripción
i_Clk	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia de 100 MHz.
i_Rst	1 bit	Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.
i_Frec_de	32 bits	Entrada que referencia a la frecuencia de trabajo, debe ser un valor almacenado en memoria.

_trabajo		
o_Timming	1 bit	Salida de referencia en el tiempo, debe ser una señal periódica con una frecuencia definida por la entrada de frecuencia de trabajo.

Tabla 4: Entradas y salidas del preescalador

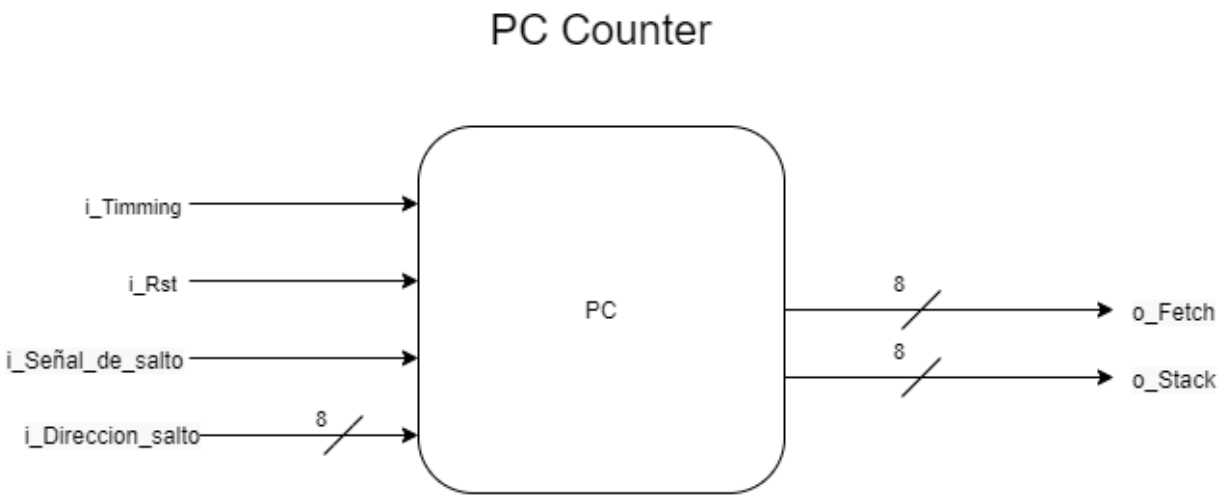

Diagrama 4: PC Counter

Entradas y salidas

Señal	Núm. de bits	Descripción
i_Timming	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia se determina por el preescalador.
i_Rst	1 bit	Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.

i_Señal_de_salto	1 bit	Entrada que indica el salto de instrucción desde la unidad de control.
i_Direccion_salto	8 bits	Contiene el valor de la dirección de la instrucción a la que se desea saltar, ya sea condicional o incondicional, o retorna al estado previo al salto almacenado en el stack.
o_Fetch	8 bits	Salida que contiene la dirección de la instrucción que se desea buscar en la memoria de instrucciones.
o_Stack	8 bits	Guarda la dirección en los registros antes de realizar un salto.

Tabla 5: Entradas y salidas del PC

Diagrama 5: Registro de instrucciones

Entradas y salidas

Señal	Núm. de bits	Descripción

i_Timming	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia se determina por el preescalador.
i_Rst	1 bit	Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.
i_Instruccion	9 bits	Entrada que transporta la instrucción desde la memoria de instrucciones.
o_Instruction _code	3 bits	Salida que contiene la parte de la instrucción (Sin los operandos) correspondiente al set de instrucciones.
o_Operandos	6 bits	Contiene la dirección de los datos que se operarán.
o_Direccionamiento _inmediato	8 bits	Entrada que contiene la información para el direccionamiento inmediato de los datos de la memoria de datos a través del bus de direcciones.

Tabla 6: Entradas y salidas del registro de instrucciones

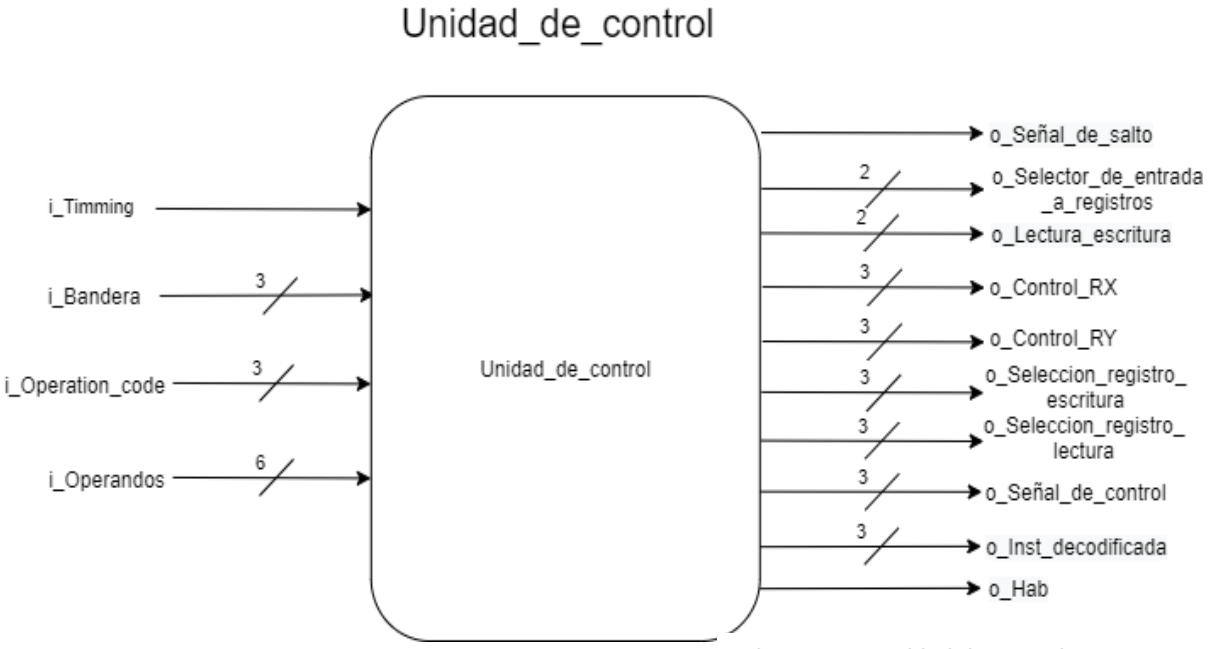

Diagrama 6: Unidad de control

Señal	Núm. de bits	Descripción
i_Timming	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia se determina por el preescalador.
i_Bandera	3 bits	Entrada que contiene las banderas de estados sobre las operaciones en la unidad lógico aritmética.
i_Operation_code	6 bits	Entrada que contiene la parte de la instrucción (Sin los operandos) correspondiente al set de instrucciones.
i_Operandos	1 bit	Entrada que contiene la dirección de los datos a operar.
o_Señal_de_salto	2 bits	Señal de control dirigida al PC counter la cual le indicar cuándo realizar un salto condicional.
o_Selector_de_ entrada_a_registros	2 bits	Salida que selecciona el dato que ingresaran a los registros de datos en el banco de registro en cada ciclo de operación. Esta va al MUX_1.
o_Lectura _escritura	3 bits	Señal de control que se encarga de indicar si se escribe o se lee un determinado dato en los registros de datos o existe un movimiento de registros.

o_Control_RX	3 bits	Se encarga de determinar qué registro se pasará como operando 1 ya sea a la ALU o al manager de salidas a memoria.
o_Control_RY	3 bits	Se encarga de determinar qué registro se pasará como operando 2 ya sea a la ALU o al manager de salidas a memoria.
o_Seleccion _registro _escritura	3 bits	Señal de gestión del movimiento de registros.
o_Seleccion _registro _lectura	3 bits	Señal de gestión del movimiento de registros.
o_Señal_de _control	3 bits	Señal de control dirigida al Manager_de_salidas_a_memoria y e indica qué datos saldrán del microprocesador a los buses de direcciones y datos de salidas a memoria.
o_Inst _decodificada	3 bits	Salida que le indica a la unidad lógico aritmética qué operación matemática o lógica realizará.
o_Hab	1 bit	Habilita a la ALU para realizar una operación.

Tabla 7: Entradas y salidas de la unidad de control

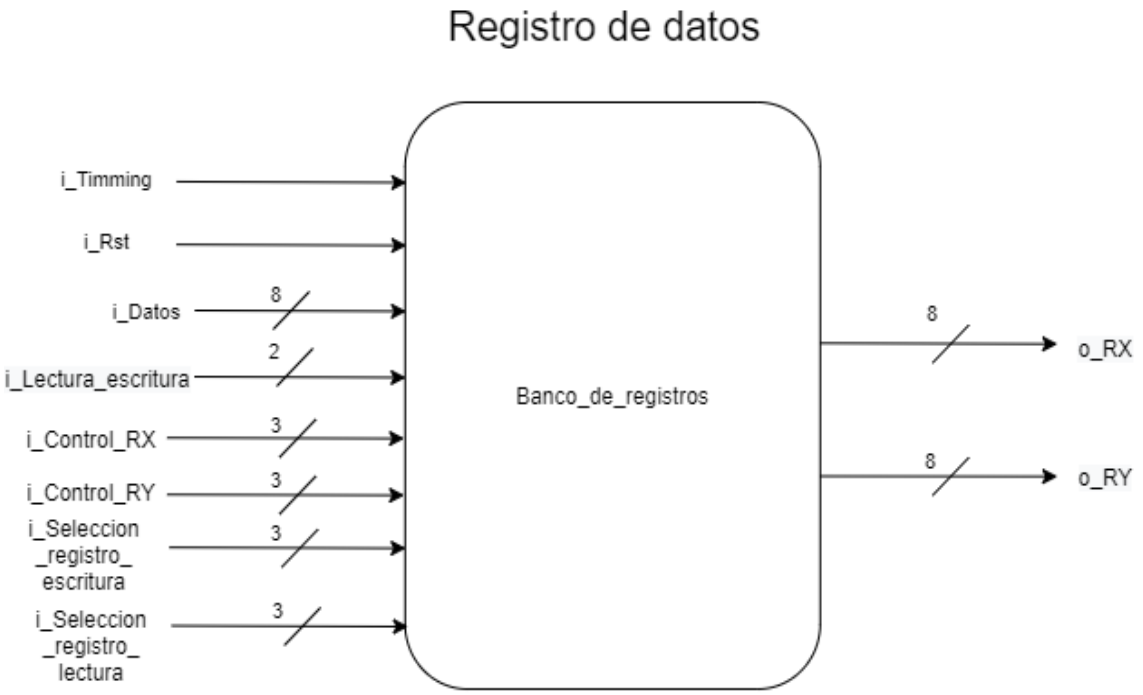

Diagrama 7: Registro de datos

Entradas y salidas

Señal	Núm. de bits	Descripción
i_Timming	1 bit	Entrada de referencia en el tiempo, debe ser una señal periódica con una frecuencia se determina por el preescalador.
i_Rst	1 bit	Entrada que restaura el sistema a una configuración inicial de los registros, memoria, etc.
i_Datos	8 bits	Contiene los datos a almacenar en los registros.
i_Lectura _escritura	2 bit	Señal de control que se encarga de indicar si se escribe o se lee un determinado dato en los registros de datos o existe un movimiento de registros.

i_Control_RX	3 bits	Se encarga de determinar qué registro se pasará como operando 1 ya sea a la ALU o al manager de salidas a memoria.
i_Control_RY	3 bits	Se encarga de determinar qué registro se pasará como operando 2 ya sea a la ALU o al manager de salidas a memoria.
i_Seleccion _registro_ escritura	3 bits	Señal de gestión del movimiento de registros.
i_Seleccion _registro_lectura	3 bits	Señal de gestión del movimiento de registros.
o_RX	8 bits	Contiene los datos del operando 1 que se procesará en la ALU o los datos ya operados que se destinarán a almacenar en la memoria de datos.
o_RY	8 bits	Contiene los datos del operando 1 que se procesará en la ALU o los datos ya operados que se destinarán a almacenar en la memoria de datos.

Tabla 8: Entradas y salidas de los registros de datos

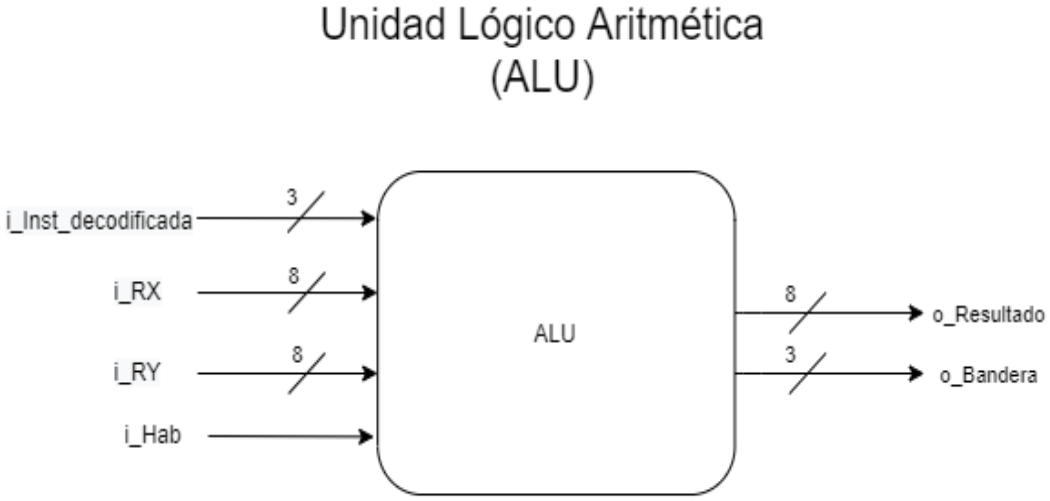

Diagrama 8: Unidad lógico aritmética

Entradas y salidas

Señal	Núm. de bits	Descripción
i_Inst _decodificada	3 bits	Entrada que le indica a la unidad qué operación matemática o lógica realizará.
i_RX	8 bitS	Entrada desde el banco de registros, contiene el operando 1 que se procesará.
i_RY	8 bits	Entrada desde el banco de registros, contiene el operando 2 que se procesará.
i_Hab	1 bit	Habilita a la ALU para realizar una operación.
o_Resultados	8 bits	Salida que contiene los datos post-procesamiento, a los cuales ya se les aplicó la operación deseada.

o_Bandera	3 bits	Salida dirigida a la unidad de control para señalar el estado de la operación procesada en la ALU, para ejecutar una acción con base a ello.

Tabla 9: Entradas y salidas de la unidad lógico arimética

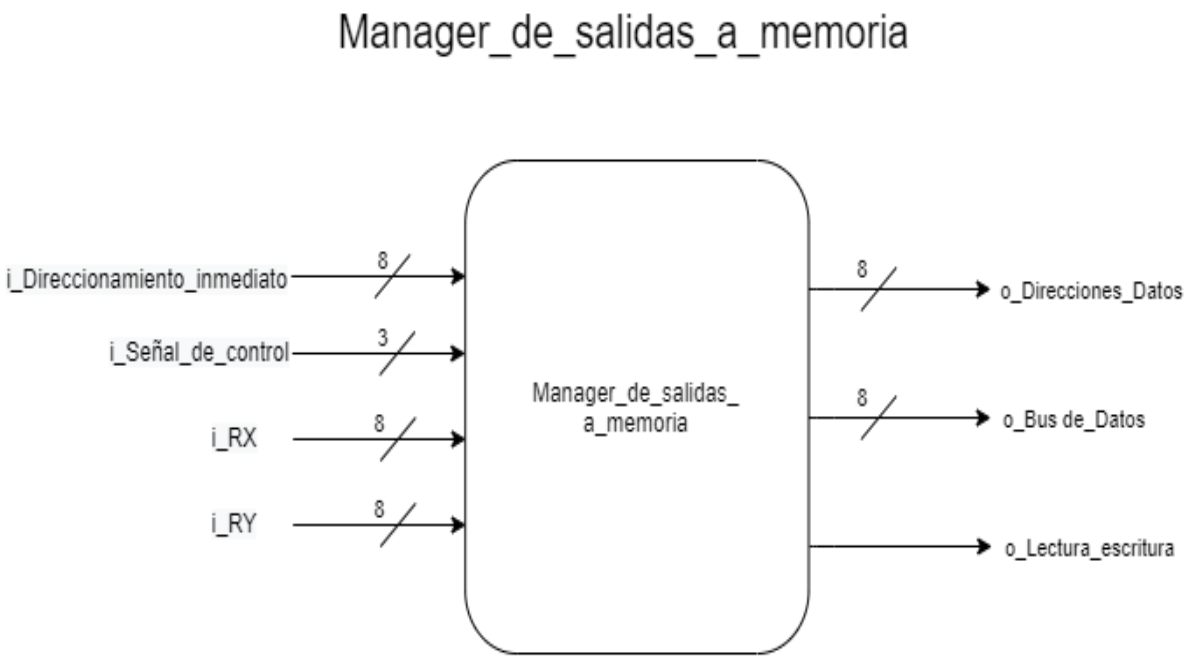

Diagrama 9: Manager de salidas a memoria de datos

Entradas y salidas

Señal	Núm. de bits	Descripción
i_Direccionamiento _inmediato	8 bits	Entrada que contiene la información para el direccionamiento inmediato de los datos de la memoria de datos a través del bus de direcciones.
i_Señal_de_control	3 bitS	Señal de control de entrada que le indica al Manager_de_salidas_a_memoria los datos que debe pasa al bus de datos de salida o al bus de direcciones.

i_RX	8 bits	Entrada que contiene la dirección de memoria en la que se escribirán los datos.
i_RY	1 bit	Entrada que contiene los datos ya procesados y preparados para su salida a memoria a través del bus de datos de salida. También puede contener la dirección de memoria de donde se cargarán los datos en caso de tener la segunda instrucción LOAD.
o_Direcciones_Datos	8 bits	Salida que se encarga de movilizar las direcciones de memoria según la operación que se realice, ya sea la función de store o load.
o_Bus_Datos	3 bits	Bus de datos de salida que moviliza los datos ya procesados para su carga o almacenaje en memoria.
o_Lectura_escritura		Le señala a la memoria de datos si se va a leer un dato o se va a escribir sobre ella.

Tabla 10: Entradas y salidas del manager de salidas a memoria

SIMULACIÓN

-Microprocesador_Legion_H

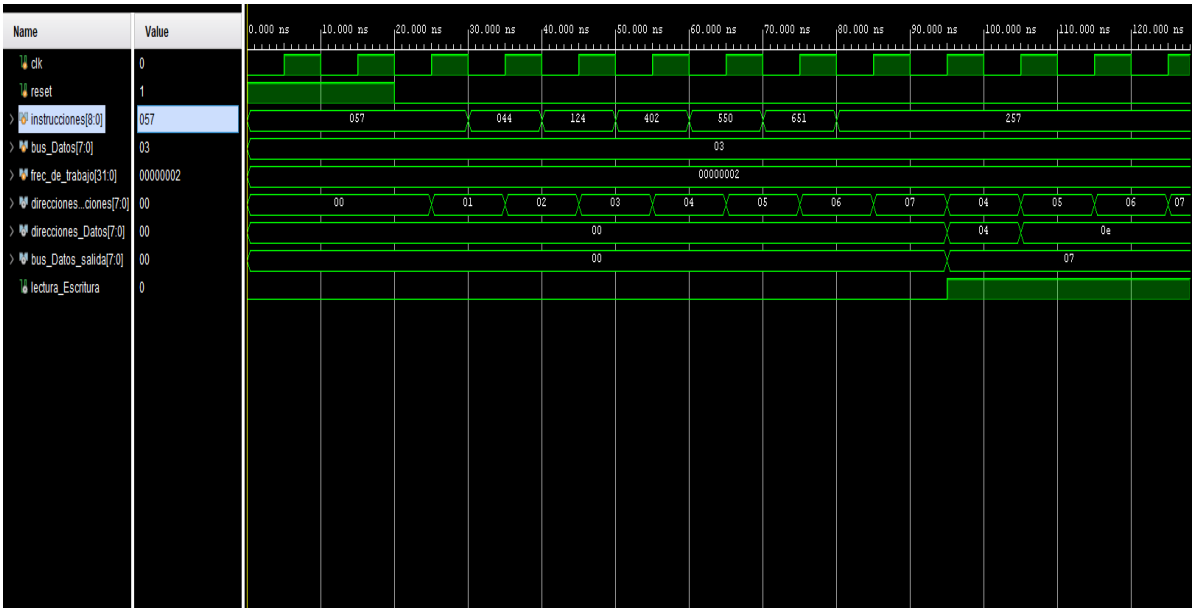

Ilustración 1: Simulación del código TOP del microprocesador

Con la finalidad de realizar pruebas de comportamiento del microprocesador, se realizó un Testbench el cual simulaba un pequeño programa almacenado, el cual consta de 7 instrucciones las cuales son las siguientes:

1. LOAD1,[101],111: Carga inmediata del número R7 en el registro R6.
2. LOAD1,[100],100: Carga inmediata del número R5 en el registro R5.
3. LOAD2,010,[100]: Carga desde la memoria de datos el dato almacenado en la dirección guardada en el registro R5 y almacénalo en el registro R3.
4. MOVE,000,010: Mueve el dato del registro R3 al registro R0.
5. MATH,101,000: Suma el dato en el registro R6 con el dato en el registro R0.
6. JUMP,[101],001: Salto incondicional a la instrucción almacenada en el registro R6 y guarda el valor del PC en el registro R7.
7. STORE,[010],111: Almacena en memoria en la dirección almacenada en el registro R3 el dato contenido en el registro R7.

Los resultados y las señales obtenidas en dicha prueba se muestran en la ilustración 1.

-Preescalador

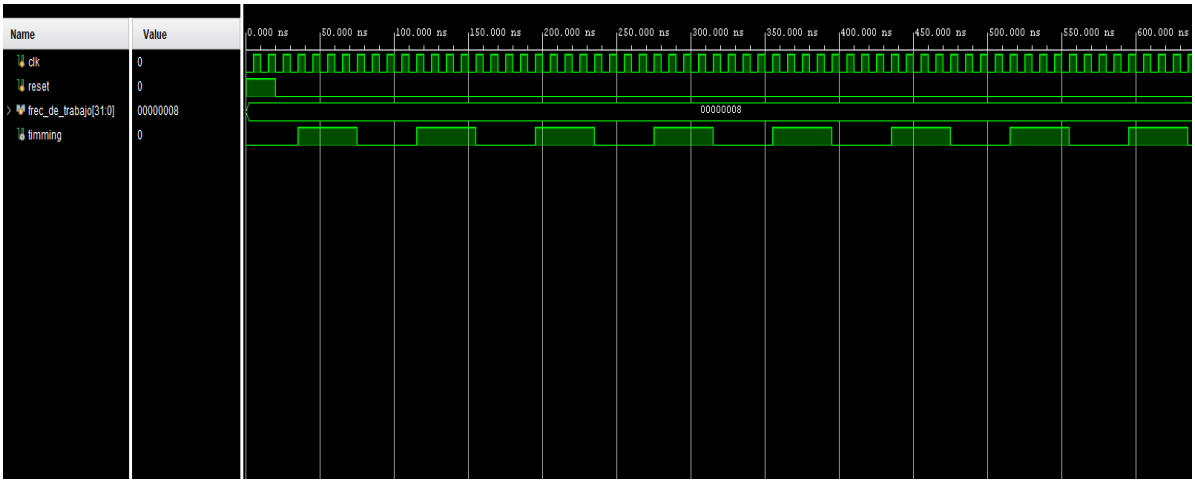

Ilustración 2: Simulación del preescalador a 25Mhz

El análisis del preescalador es sencillo, en la ilustración 2 se puede ver la variación del reloj interno de la basys 3 el cual se encuentra a 100Mhz y la señal de salida como un '1' lógico cada 4 oscilaciones de reloj, lo que correspondería una división de frecuencia de 100Mhz a 25 MHz que para los fines de la práctica representa la frecuencia variable establecida por la entrada i_Frec_de_trabajo cada que se establece un reset en el microprocesador, la cual en este caso es de 4, este componente tiene la finalidad de variar la frecuencia de funcionamiento del microprocesador para fines de análisis y pruebas.

-PC Counter

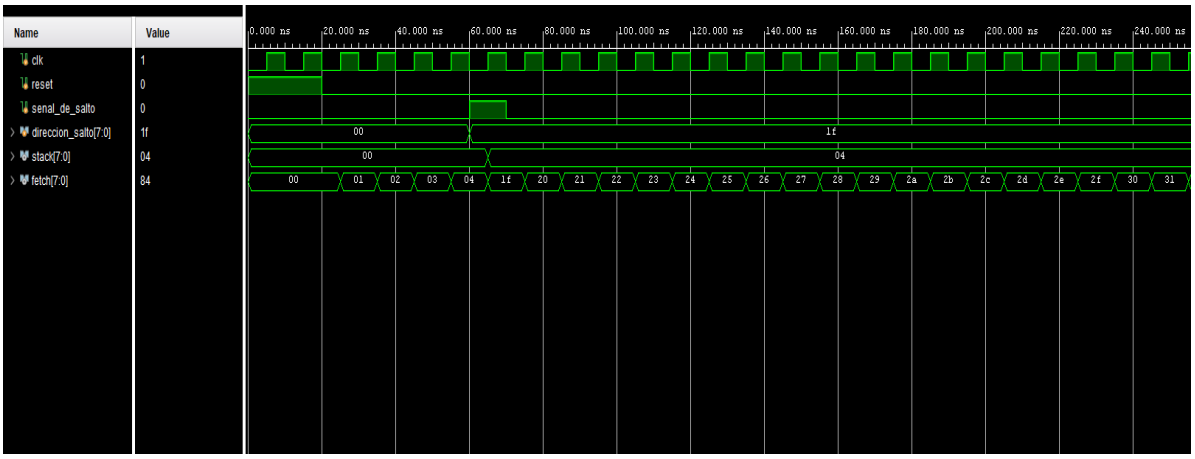

Ilustración 3: Simulación del PC Counter

Como se puede apreciar en la ilustración 3, el contador del sistema o PC se autoincrementa con cada ciclo de rejo, lo que se traduce en el incremento de la instrucción a solicitar a través del bus de direcciones de instrucciones durante el proceso de fetching. Como se puede notar, este componente es capaz de ejecutar saltos entre instrucciones y enviar a almacenar la dirección de la instrucción en la que se quedó antes de realizar el salto, cuando la señal de i_Señal_de_salto se encuentra en alto, la salida o_Señal_a_stack pasa a tener el valor anterior del contador y la salida o_Fetch slata a la dirección que hay en la entrada i_Direccion _salto.

-Registro de instrucciones

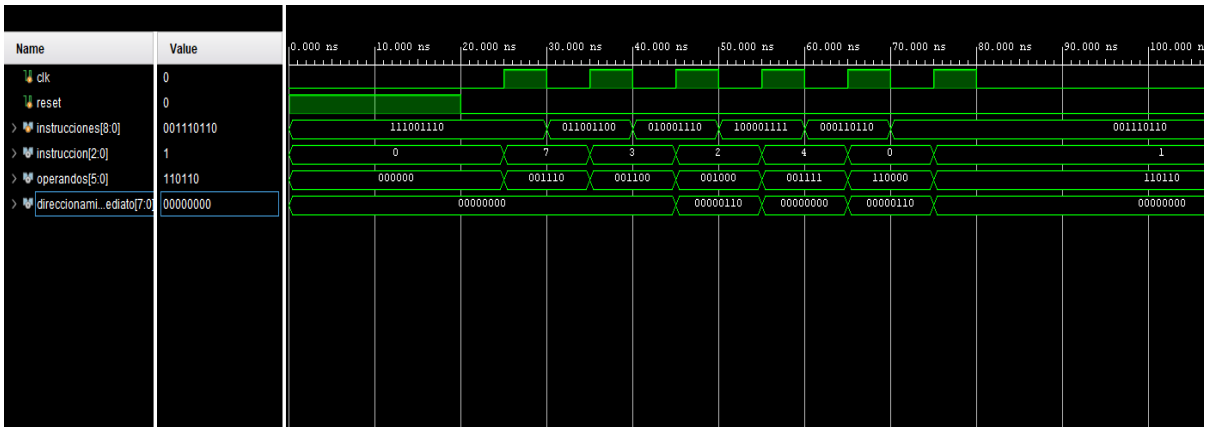

Ilustración 4: Simulación del registro de instrucciones

En la ilustración 4 vemos el comportamiento del registro de instrucciones, el cual decodifica almacena temporalmente la instrucción a ejecutar proveniente de la entrada i_Instrucciones y la decodifica para su posterior uso por parte de la Unidad de control. Como se puede observar, este componente descompone la instrucción en 2 partes, el operation code representado por la salida o_Instruccion a la que corresponden los primeros 3 bits desde el más significativo y los operandos representados en la salida o_Operandos a la que corresponden los 6 bits restantes de la instrucción, en caso de recibir las instrucciones LOAD 1 o STORE 1 las cuales corresponden al direccionamiento inmediato de un dato, asigna los bits [5:3] de la instrucción a o_Direccionamiento_inmediato y los bits [2:0] a la parte más significativa de o_Operandos, rellenando los demás bits con 0's.

-Registro de datos

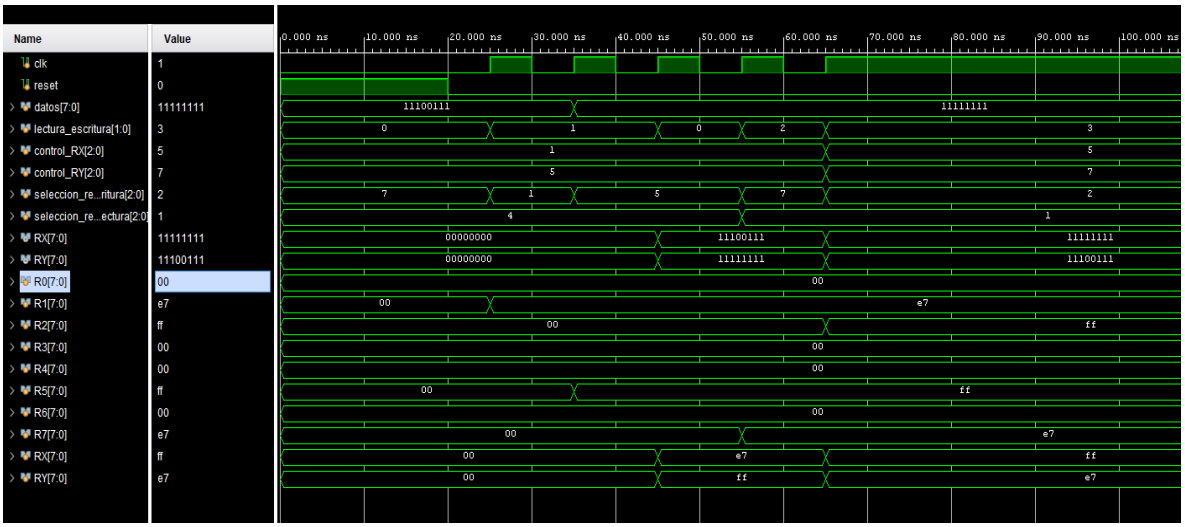

Ilustración 5: Simulación del registro de instrucciones

En la ilustración 5 podemos ver las entradas de prueba para realizar los procesos de lectura, escritura y gestión de registros, también se incluye la variación y modificación de los registros internos dependiendo de la operación a realizar. Se puede observar que la primera prueba es un proceso de escritura representado por un 1 DEC en la entrada i_Lectura_escritura, se trata de una escritura de los datos de entrada en el registro R1, la segunda prueba es también una escritura, esta se realiza en el registro R5.

La tercera prueba es una lectura de registros representado por un 2 DEC en la entrada i_Lectura_escritura, se trata de pasar el contenido del registro R1 a o_RX según la entrada i_Control_RX y el contenido del registro R5 a o_RY según la entrada i_Control_RY.

La cuarta prueba es un movimiento de registros representado por un 3 DEC en la entrada i_Lectura_escritura, se trata del movimiento de los datos del registro R1 al registro R7 replicando el valor e7 HEX que este tenía.

Por último, la prueba de lectura y escritura simultánea representado por un 4 DEC en la entrada i_Lectura_escritura, se pasa el valor de los registros R5 y R7 a o_RX y o_RY respectivamente, así como la escritura del dato entrante de i_Datos en el registro R2, cambio que se ve reflejado en el valor de la señal interna de este.

-Unidad de control

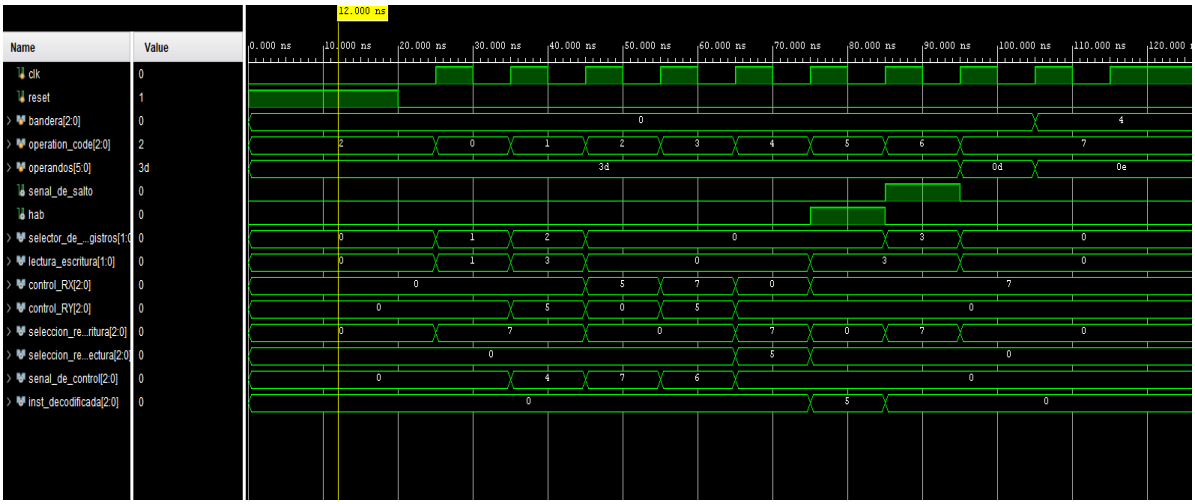

Ilustración 6: Simulación de la unidad de control

En la ilustración 6 podemos notar que se realiza una prueba con las entradas i_Operandos elegida de manera aleatoria y se observan las salidas para todas y cada una de las instrucciones de nuestro set de instrucciones.

-Unidad lógico aritmética (ALU)

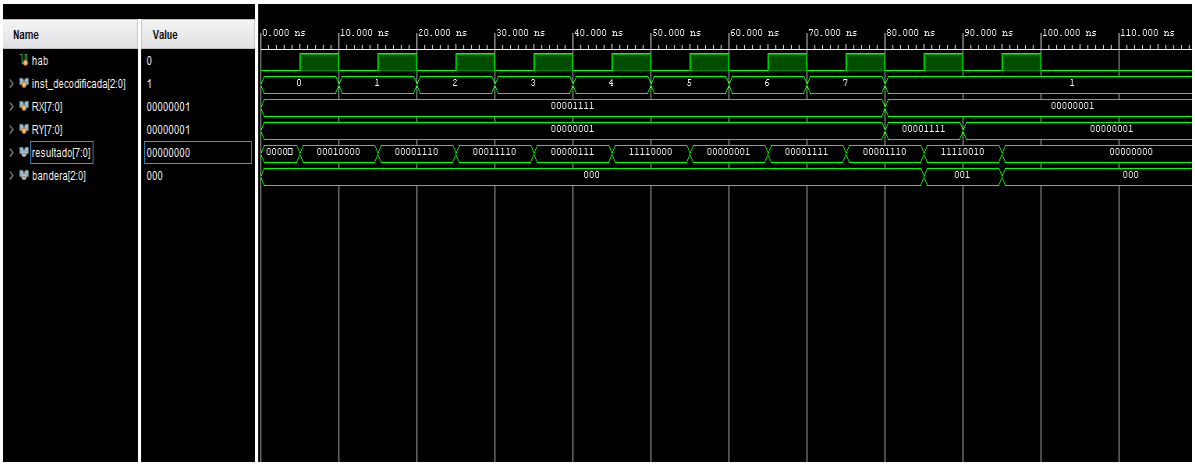

Ilustración 7: Simulación de la unidad lógico aritmética

Como se observa en la ilustración 7 se realizaron las pruebas de todas las operaciones que es posible realizar en la unidad lógico aritmética, los valores de prueba fueron 00001111 para i_RX y 00000001 para i_RY, como podemos ver, en la primera operación que es la suma, el resultado de dicha operación es 00010000 el cual es correcto debido a que simplemente

es la suma de un bit al dato de i_RX, la resta da como resultado 00001110, el corrimiento a la izquierda da como resultado 00001110 debido a que solamente se recorre una posición el valor de i_RX, el corrimiento a la derecha da como resultado 00000111 por las mismas razones de la prueba anterior, la negación da como resultado 11110000, debido a que se asigna un 1 a todos los valores 0 de i_RX y un 0 a todos sus valores 1, el resultado de la operación lógica AND da como resultado 00000001 debido a que el único bit en el que i_RX e i_RY coinciden en alto es el primer bit, el resultado de la operación lógica OR es 00001111 y finalmente el resultado de la potenciación es 00001110.

-Manager de salidas a memoria

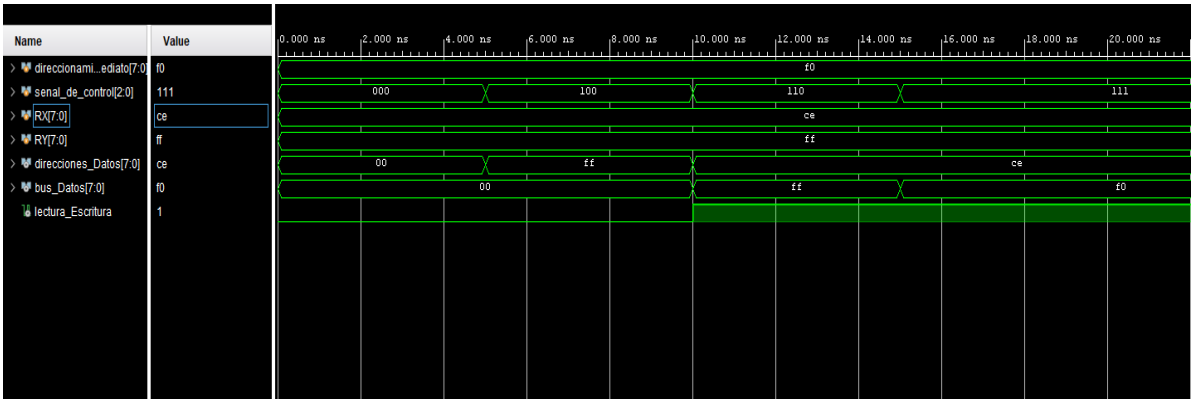

Ilustración 8: Simulación del manager de salidas a memoria

En la ilustración 8 las salidas del Manager_de_salidas_a_memoria dependiendo de la entrada i_Señal_de_control, la cual se codifica de la sig. manera: El bit [2] se encarga de la habilitación del componente, esto debido a que es un componente combinacional, el bit [1] se encarga de establecer si se realiza un proceso de lectura o escritura en memoria y el bit menos significativo solamente señala si se hará un direccionamiento inmediato o indirecto a través de los registros.

La primera prueba consta de una lectura de memoria, estableciendo como dirección el valor de la entrada i_RY, la segunda prueba es una escritura en memoria por direccionamiento indirecto, pasando el valor de i_RX a o_Direcciones_datos y el valor de i_RY a o_Bus_Datos; la última prueba se trata de una escritura con direccionamiento inmediato, pasando el valor de i_Direccionamiento_inmediato a o_Bus_Datos y el valor de i_RX a o_Direcciones_datos.

-Multiplexor de entrada a registros de datos (MUX)

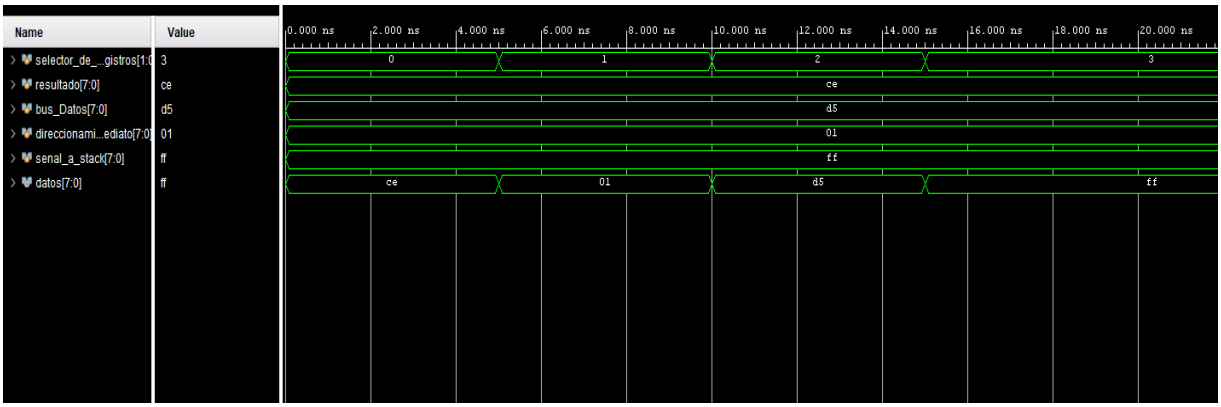

Ilustración 9: Simulación del multiplexor de entrada a registros de datos

En la ilustración 9 podemos apreciar el funcionamiento del módulo MUX, el cual se encarga de seleccionar el dato que ingresará al Registro_de_datos, este componente es muy sencillo de analizar, selecciona el valor a pasar a su salida dependiendo de la entrada i_Selector_de_entrada_a_resgistros. En la primera prueba el selector toma el valor de 00 BIN por lo que la salida o_Datos adquiere el valor de la entrada i_Resultado el cual es CE HEX, en la segunda prueba el selector toma el valor de 01 BIN por lo que la salida o_Datos adquiere el valor de la entrada i_Direccionamiento_inmediato el cual es 01 HEX, en la tercera prueba el selector toma el valor de 10 BIN por lo que la salida o_Datos adquiere el valor de la entrada i_Bus_Datos el cual es D5 HEX y finalmente en la última prueba el selector toma el valor de 11 BIN por lo que la salida o_Datos adquiere el valor de la entrada i_Señal_a_stack el cual es FF HEX.

IMPLEMENTACIÓN

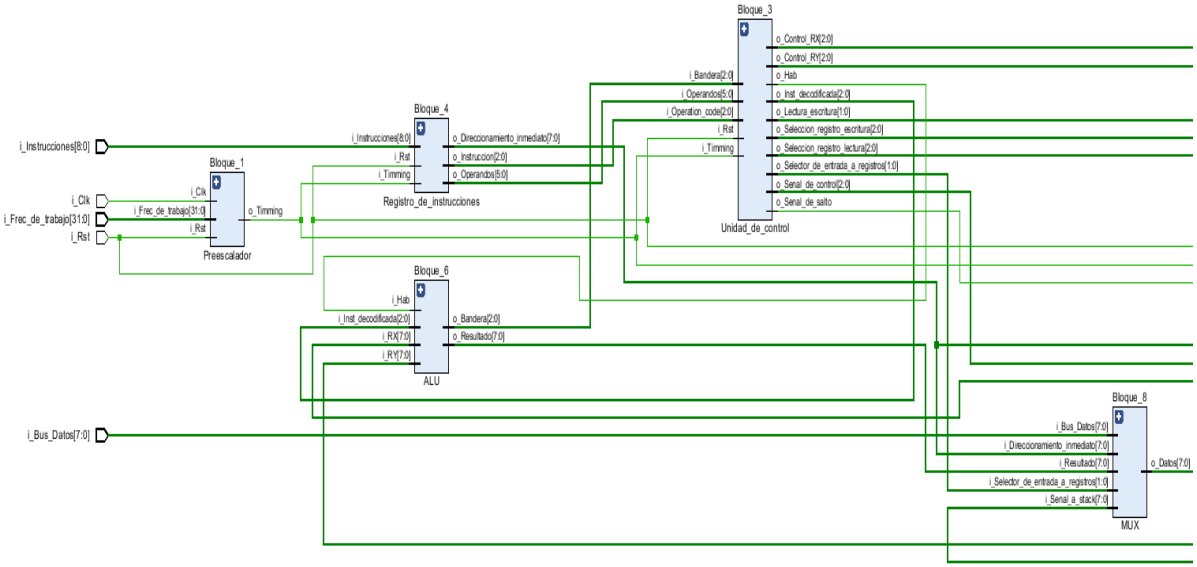

Ilustración 10: Diagrama RTL del microprocesador (Parte 1)

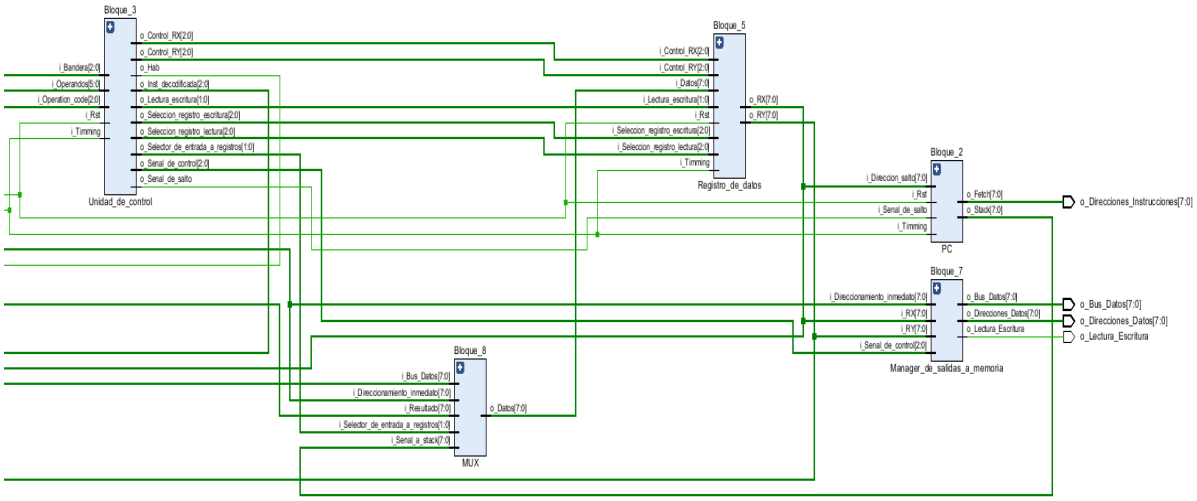

Ilustración 11: Diagrama RTL del microprocesador (Parte 2)

El esquema RTL mostrado en las ilustraciones 9 y 10 coincide en gran medida con el diagrama de caja blanca de nuestro dispositivo, las conexiones realizadas en el código TOP del proyecto entre los componentes según nuestro esquema RTL son las siguientes: La salida o_Timming del preescalador variable se conecta a la entrada de reloj i_Clk de los

componentes secuenciales PC, Registro_de_instrucciones, Registro_de_datos y a la Unidad_de_control.

La salida de o_Resultado de la ALU, la entrada del microprocesador i_Datos, la salida o_Señal_a_stack del PC y salida de o_Direccionamiento_inmediato del Registro_de_instrucciones se multiplexan en el MUX a través de la señal Selector_de_entrada_a_registros provenientes de la Unidad_de_control para su ingreso como una señal única i_Datos en el Registro_de_datos.

La salida o_Banderas de la ALU se conecta a la entrada i_Banderas de la Unidad_de_Control, con el fin de que esta, evalúe la condición de esta señal para la ejecución de saltos de instrucciones a través de la salida o_Señal_de_salto que se conecta a la entrada i_Señal_de_salto del PC.

Las salidas o_RX y o_RY del Registro_de_datos se conectan a los componentes combinacionales ALU y Manager de salidas a memoria, a la primera con el fin de operar los datos que estas contienen, la segunda para el proceso de almacenamiento y direccionamiento de datos a la memoria, por último, la salida i_RY se conecta a la entrada i_Direccion_de_salto del PC con el fin de brindarle la dirección de la instrucción a la que saltará en caso de que i_Señal_de_salto se encuentre en alto.

La salida o_Hab de la unidad de control se conecta a la entrada o_Hab de la ALU para habilitar su funcionamiento, debido a que es un circuito combinacional, la señal Inst_decodificada se conecta a la entrada i_Inst_decodificada de la ALU y le señala la operación a realizar en caso de que la señal de habilitación de esta se encuentre en alto.

La Unidad_de_control logra gestiona los procesos de lectura y escritura del Registro_de_datos a partir de las señales Control_RX (La cual controla el valor que se pasará a la salida RX), Control_RY (La cual controla el valor que se pasará a la salida RY), Lectura_escritura (Establece el proceso a realizar en registros, puede ser de lectura, de escritura, un movimiento de registros o lectura y escritura simultáneas), Selector_registro_lectura (Para seleccionar el registro que se leerá en caso de un movimiento de registros) y Selector_registro_escritura (Para seleccionar el registro que se escribirá en caso de un movimiento de registros o de un proceso de lectura-escritura simultáneos).

La salida o_Direccionamiento_inmediato del Registro_de_instrucciones se conecta a la entrada i_Direccionamiento_inmediato del Manager_de_salidas_a_memoria y a la entrada 01 del multiplexor MUX, se encarga de llevar el dato #NUM en caso de la ejecución de las instrucciones LOAD 1 y STORE 1 del set de instrucciones, las cuales, hacen referencia a la carga inmediata de un dato en registros y en memoria, respectivamente.

Las salidas del Manager_de_salidas_a_memoria o_Direcciones_Datos, o_Lectura_Escritura y o_Bus_Datos se conectan con sus respectivas contrapartes en las salidas del microprocesador, se encargan del direccionamiento de memoria, la selección de escritura o lectura el paso del dato a escribir en memoria respectivamente.

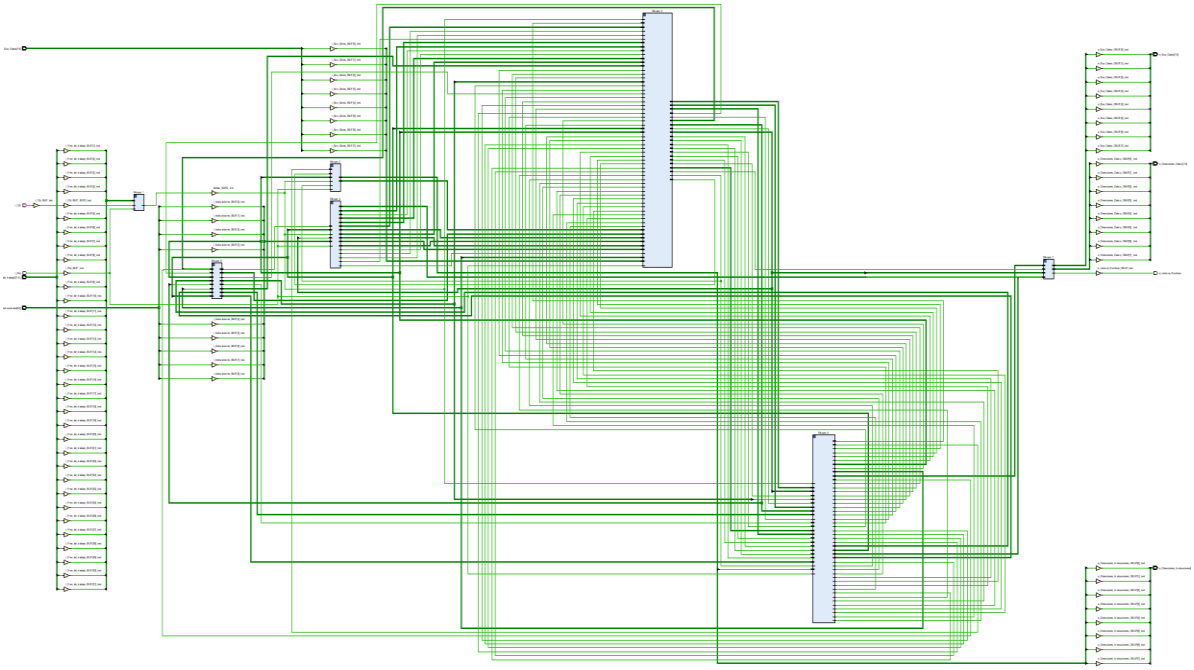

Ilustración 12: Diagrama esquemático de la implementación del dispositivo

En la ilustración 12 se observa el diagrama del circuito implementado y en la ilustración 13 las características generales de la implementación del proyecto, se usó el 1% de las LUTS, el 27% de las entradas y salidas, el 1% de los flip flops y el 3% de los BUFG de la Basys 3.

En lo que a desempeño energético se refiere, los cálculos realizados por el software fueron los siguientes: Un desempeño de 26.405 Watts, una temperatura de funcionamiento de 125.0°C con un margen térmico de -57.0°C, estos datos de muestran más a fondo en las ilustraciones 14 y15.

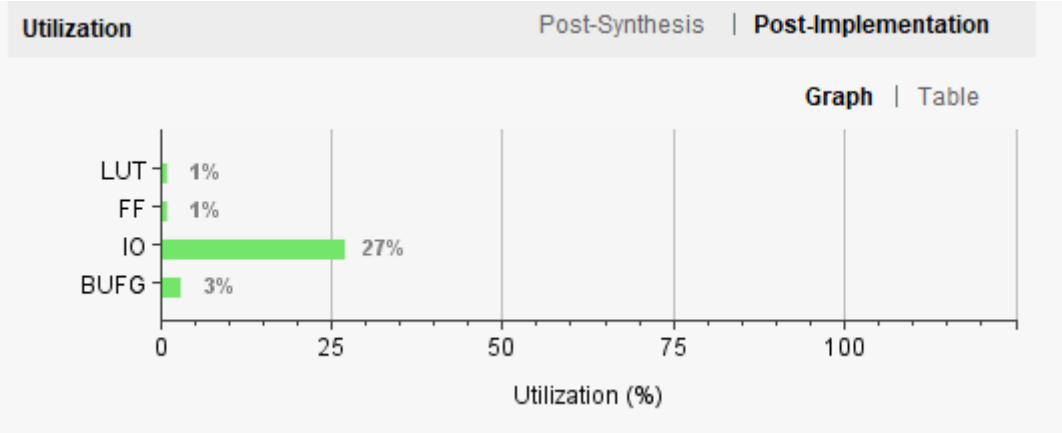

Ilustración 13: Tabla de utilización post implementación

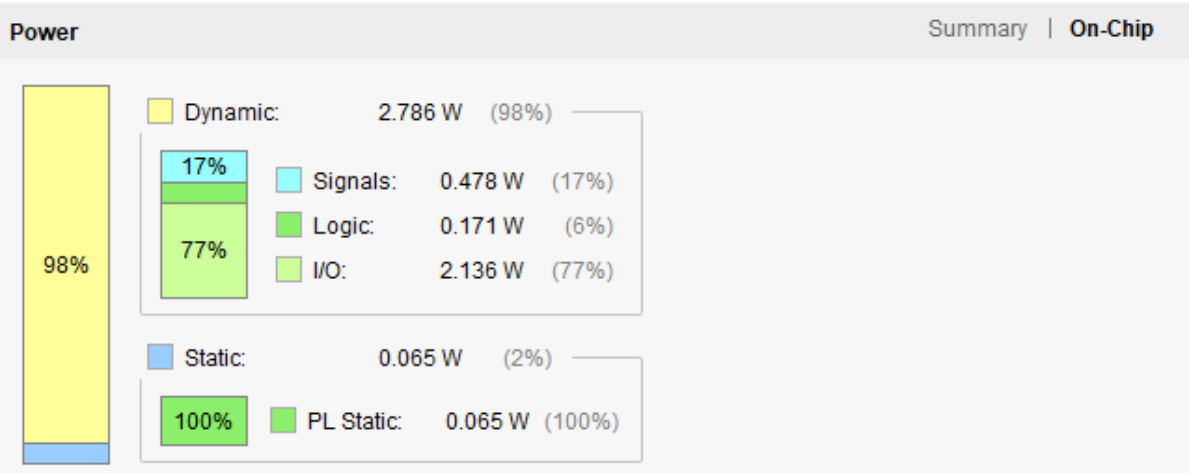

Ilustración 14: Consumo energético del chip

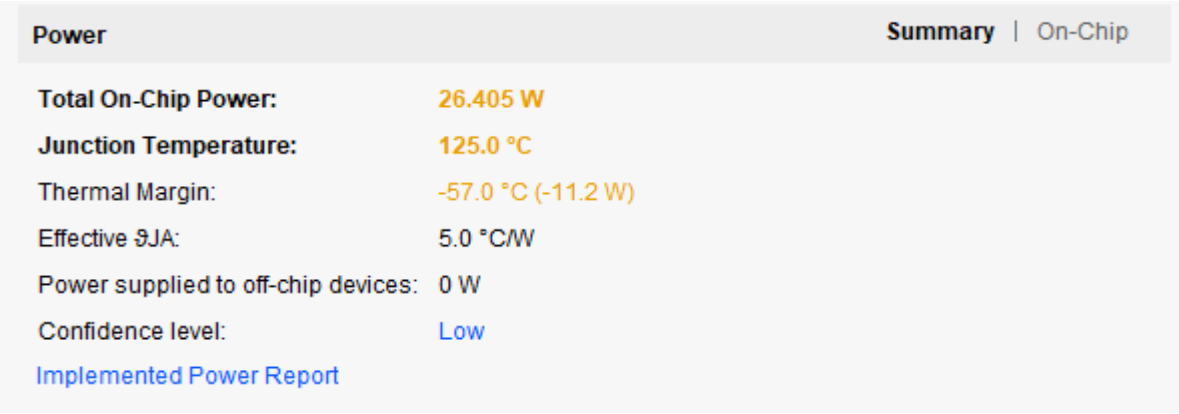

Ilustración 15: Resumen de consumo energético

CONCLUSIONES

Durante el desarrollo de este proyecto se tuvieron distintas interrogantes acerca del proceso de diseño de un microprocesador, pero gracias a la metodología de análisis y diseño Top-Down, se procedió de la parte más general que era el propósito del microprocesador como entidad única, hasta la división de procesos entre distintos componentes internos destinados a realizar tareas específicas, pero que, al formar parte de un sistema, logran interactuar entre sí y obtener el resultado deseado.

El microprocesador diseñado cumple con los parámetros establecidos al inicio del proyecto, posee un set de instrucciones reducido (RISC) y una arquitectura que le permite agilizar procesos y realizar operaciones complicadas cuyo abanico de soluciones es muy amplio, subdividiéndolas en otras más sencillas las cuales poseen una sola solución, despejando así la carga de trabajo en el microprocesador y mejorando el desempeño energético, a costa de un mayor número de instrucciones a realizar por cada operación.

El uso de una arquitectura Harvard para el diseño de un microprocesador permite diferir entre la memoria de instrucciones y la memoria de datos, agilizando los procesos de fetch y execute permitiendo una verdadera paralelización de tareas, aspecto que a pesar de emularse en arquitecturas Princeton, solamente es posible en la Harvard. En dicha arquitectura todas las operaciones se realizan en registros y no en la memoria, permitiendo una mayor velocidad de comunicación debido a la construcción de la memoria de registros dentro del microprocesador, la cual es mucho más rápida que la memoria de acceso aleatorio desde la que se cargan los datos, optimizando los tiempos de operación.

Actualmente la tendencia de la portabilidad de los dispositivos, supone un reto para el desarrollo de microprocesadores, los cuales deben ser más compactos, más eficientes y por supuesto, más veloces, aspecto que se logra con el uso de la arquitectura RISC, también aplicada a lo largo de este proyecto, esta arquitectura se propone desplazar casi por completo a las arquitecturas CISC x86 y x64, debido a su gran desempeño en dispositivos

móviles además de la ya mencionada multitarea y a la división de las tareas y operaciones en varios núcleos del procesador.

Habiendo tenido este acercamiento al funcionamiento, diseño y operación de los microprocesadores y de las distintas arquitecturas de computadoras, esperamos en un futuro, ser capaces de proponer soluciones que involucren la implementación de sistemas de control basados en microcontroladores con el fin de atacar diferentes problemáticas en el campo de la industria, la ingeniería y la investigación, desafíos como el internet en las cosas, la inteligencia artificial, la monitorización de la salud, entre otras áreas, teniendo en cuenta las necesidades y objetivos del mundo de la tecnología y sobre todo, de la sociedad globalizada en la que vivimos.

REFERENCIAS

[1] Autores de Wikipedia (2018, febrero 28) Microprocesador [En línea] Disponible en: www.wikipedia.org

[2] Autor anónimo (2019, marzo 07) CISC, RISC, VLIW y EPIC [En línea] Disponible en: www.ieec.uded.es

[3] René Camacho (2012, abril 9) Arquitectura von Neumann y arquitectura Harvard [En línea] Disponible en: www.rcmcomputointegrado.blogspot

[4] Helmut Corvo (2019) Unidad de control [En línea] Disponible en: www.lifeder.com

[5] Saifuddin Abdullah (2020, junio 12) Registros de un microprocesador [En línea] Disponible en: www.techlandia.com

[6] Autores de Wikipedia (2020, agosto 07) Conjunto de instrucciones [En línea] Disponible en: www.wikipedia.org

[7] Josep Roca (2018, septiembre 21) ISA, ARM y X86, set de instrucciones de una CPU [En línea] Disponible en: Sitio web: www.hardzone.es

APÉNDICE A (Códigos)

-Microprocesador Legion H

```
//La finalidad de este modulo es reunir los componentes e interconectarlos
//para crear el sistema final, asi como agregar sus entradas y salidas
//Funge como parte top del sistema
module Microprocesador_Legion_H
    (   input i_Clk,
        input i_Rst,
        input [8:0] i_Instrucciones,
        input [7:0] i_Bus_Datos,
        input [31:0] i_Frec_de_trabajo,
        output [7:0] o_Direcciones_Instrucciones,
        output [7:0] o_Direcciones_Datos,
        output [7:0] o_Bus_Datos,
        output o_Lectura_Escritura);


    wire Timming;
    wire Hab;
    wire Senal_de_salto;
    wire [7:0] Senal_a_stack;
    wire [7:0] Direccionamiento_inmediato;
    wire [5:0] Operandos;
    wire [2:0] Operation_code;
    wire [2:0] Senal_de_control;
    wire [7:0] RX;
    wire [7:0] RY;
    wire [7:0] Resultado;
    wire [2:0] Bandera;
    wire [7:0] Datos;
    wire [1:0] Lectura_escritura;
    wire [2:0] Control_RX;
    wire [2:0] Control_RY;
    wire [2:0] Seleccion_registro_escritura;
    wire [2:0] Seleccion_registro_lectura;
    wire [2:0] Inst_decodificada;
    wire [1:0] Selector_de_entrada_a_registros;
    // Se declaran las señales internas que conectaran nuestros componentes

    //Preescalador
    Preescalador Bloque_1(
        .i_Clk(i_Clk),
```

```
        .i_Rst(i_Rst),
        .i_Frec_de_trabajo(i_Frec_de_trabajo),
        .o_Timming(Timming)
    );
    //Se encarga de dividir la frecuencia de entrada para adaptarla a la
    //frecuencia de funcionamiento de los componentes internos, otorga una
    //señal lógica cuyo fin será sincronizar los procesos del sistema.
    //De frecuencia variable determinada por i_Frec_de_trabajo


    //PC Counter
    PC Bloque_2(
        .i_Timming(Timming),
        .i_Rst(i_Rst),
        .i_Senal_de_salto(Senal_de_salto),
        .i_Direccion_salto(RX),
        .o_Stack(Senal_a_stack),
        .o_Fetch(o_Direcciones_Instrucciones)
    );
    //El PC counter que funge como puntero de instrucciones, se encarga de
    //registrar e indicar la posición del procesador en la secuencia de
    //instrucciones e incrementa con cada ciclo de la señal timming, en caso
    //de romper con la secuencia se ayuda con un registro de stack o pila par
a
    //guardar la posición en la que se encontraba para posteriormente volver
a
    //ella.


    //Unidad de control
    Unidad_de_control Bloque_3(
        .i_Timming(Timming),
        .i_Rst(i_Rst),
        .i_Bandera(Bandera),
        .i_Operation_code(Operation_code),
        .i_Operandos(Operandos),
        .o_Senal_de_salto(Senal_de_salto),
        .o_Selector_de_entrada_a_registros(Selector_de_entrada_a_registros),
        .o_Lectura_escritura(Lectura_escritura),
        .o_Control_RX(Control_RX),
        .o_Control_RY(Control_RY),
        .o_Seleccion_registro_escritura(Seleccion_registro_escritura),
        .o_Seleccion_registro_lectura(Seleccion_registro_lectura),
        .o_Senal_de_control(Senal_de_control),
        .o_Inst_decodificada(Inst_decodificada),
        .o_Hab(Hab)
    );
```

```
    //Decodifica las instrucciones que llegan desde el registro de instruccio
nes
    //y actúa conforme al set de instrucciones establecido y coordina las ope
raciones
    //en la unidad lógica aritmética, también analiza el estado de las bander
as y
    //conforme a este establece un salto o no en el PC counter o envía una se
ñal para
    //que se pase algún valor almacenado en el registro al bus de datos para
su posterior
    //almacenamiento en memoria


    //Registro de instrucciones
    Registro_de_instrucciones Bloque_4(
        .i_Timming(Timming),
        .i_Rst(i_Rst),
        .i_Instrucciones(i_Instrucciones),
        .o_Instruccion(Operation_code),
        .o_Operandos(Operandos),
        .o_Direccionamiento_inmediato(Direccionamiento_inmediato)
    );
    //Este componente permite la decodificación y almacenamiento de la instru
cción
    //a ejecutar, la cual proviene del bus de la memoria de instrucciones, ta
mbién
    //permite la función de direccionamiento inmediato. Es un registro.


    //Registro de datos
    Registro_de_datos Bloque_5(
        .i_Timming(Timming),
        .i_Rst(i_Rst),
        .i_Datos(Datos),
        .i_Lectura_escritura(Lectura_escritura),
        .i_Control_RX(Control_RX),
        .i_Control_RY(Control_RY),
        .i_Seleccion_registro_escritura(Seleccion_registro_escritura),
        .i_Seleccion_registro_lectura(Seleccion_registro_lectura),
        .o_RX(RX),
        .o_RY(RY)
    );
    //Este componente permite el almacenamiento de los datos provenientes de
la
    //memoria de datos, guarda los resultados de las operaciones realizadas p
or
    //la ALU y guarda la dirección previa al salto del PC. Es un registro.
```

```
    //Unidad Lógico Aritmética
    ALU Bloque_6(
        .i_Inst_decodificada(Inst_decodificada),
        .i_RX(RX),
        .i_RY(RY),
        .i_Hab(Hab),
        .o_Resultado(Resultado),
        .o_Bandera(Bandera)
    );

    //Manager de salidas a memoria
    Manager_de_salidas_a_memoria Bloque_7(
        .i_Direccionamiento_inmediato(Direccionamiento_inmediato),
        .i_Senal_de_control(Senal_de_control),
        .i_RX(RX),
        .i_RY(RY),
        .o_Direcciones_Datos(o_Direcciones_Datos),
        .o_Bus_Datos(o_Bus_Datos),
        .o_Lectura_Escritura(o_Lectura_Escritura)
    );
    //Este componente permite seleccionar los datos que se pasarán al bus de
datos
    //de salida a memoria, al bus de direccionamiento y si se realizará un pr
oceso
    //de lectura o escritura en memoria. Es un componente meramente combinaci
onal.


    //Multiplexor
    MUX Bloque_8(
        .i_Selector_de_entrada_a_registros(Selector_de_entrada_a_registros),
        .i_Resultado(Resultado),
        .i_Direccionamiento_inmediato(Direccionamiento_inmediato),
        .i_Bus_Datos(i_Bus_Datos),
        .i_Senal_a_stack(Senal_a_stack),
        .o_Datos(Datos)
    );
    //Este componente permite seleccionar el dato que entrará a los
    //registros de datos. Es meramente combinacional.

endmodule
```

-Prescalador

```
//Este componente sirve como un divisor de frecuencia
//de la señal de reloj interna de 100Mhz de la basys 3 a 120Hz que
//es la frecuencia que se requiere para desplegar cada digito en el
//display, se podria decir que cada 2.4Micro segundos otorga un pulso
//positivo que dura la misma cantidad de tiempo y posteriormente
//baja a un valor logico '0'
  module Preescalador(
    input i_Clk,
    input i_Rst,
    output o_Presc);

    //Se establece un registro para el contador, que debe almacenar un valor
    //de hasta 833,333 iteraciones que es lo equivalente al periodo requerido
    reg [19:0] contador;
    //Un registro salida para trabajar en el always y que su valor a la salid
a
    //o_Presc
    reg salida;

    //Se elimino el reset asincrono debido a que causaba ambiguedades en la
    //etapa de implemetacion
    always@(posedge i_Clk)
    begin
        if (i_Rst) begin//El reset es sincrono
        contador <= 0;
        end
        else begin
            contador <= contador+1;
        end
        if (contador <= 416667) begin//Se compara si el contador llega a la
                                     // Mitad del periodo para establecer un '
1'
                                     //logico en la salida, si no, lo deja en
'0'
        salida <= 0;
        end
        else begin
        salida <= 1;
        end
        if (contador == 833334)begin//Se verifica que el contador complete el
                                     //periodo para establecer la salida a 0

        contador <= 0;//Se retorna el valor del contador a 0
        end
```

```
    end
    assign o_Presc = salida;//Se pasa el valor del registro salida a o_Presc
endmodule
```

-PC Counter

```
//Este componente permitira llevar la cuenta de la instrucción que se está
//ejecutando, se autoincreenta con cada ciclo de reloj y ejecuta saltos de
//instrucciones en caso de requerirlo.

 module PC(
    input i_Timming,
    input i_Rst,
    input i_Senal_de_salto,
    input [7:0] i_Direccion_salto,
    output[7:0] o_Stack,
    output[7:0] o_Fetch);

    // Se declaran los registros y las señales internas
    reg [7:0] cont=0;
    reg [7:0] stack=0;

    //Etapa de implemetacion
    always@(posedge i_Rst, posedge i_Timming)
    begin
        if (i_Rst) begin//El reset es asincrono
        cont <= 8'b00000000;
        end
        else begin
            if (i_Senal_de_salto==1)  begin
                stack<=cont;
                cont<=i_Direccion_salto;
            end
            else begin
                cont<=cont+1;
            end
        end
    end

    assign o_Fetch = cont;//Se pasa el valor del registro cont a o_Fetch
    assign o_Stack = stack;//Se pasa el valor del registro stack a o_Stack
endmodule
```

-Unidad de control

```
//Este componente permitira el control de las demás partes del microprocesado
r
//se encarga de verificar la ejecución de las instrucciones y del paso de los
//parámetros con los que trabajarán.
module Unidad_de_control
    (   input i_Timming,
        input i_Rst,
        input [2:0] i_Bandera,
        input [2:0] i_Operation_code,
        input [5:0] i_Operandos,
        output o_Senal_de_salto,
        output [1:0] o_Selector_de_entrada_a_registros,
        output [1:0] o_Lectura_escritura,
        output [2:0] o_Control_RX,
        output [2:0] o_Control_RY,
        output [2:0] o_Seleccion_registro_escritura,
        output [2:0] o_Seleccion_registro_lectura,
        output [2:0] o_Senal_de_control,
        output [2:0] o_Inst_decodificada,
        output o_Hab);

    // Se declaran los registros y las señales internas
    reg [1:0] Selector_de_entrada_a_registros=0;
    reg [1:0] Lectura_escritura=0;
    reg [2:0] Control_RX=0;
    reg [2:0] Control_RY=0;
    reg [2:0] Seleccion_registro_escritura=0;
    reg [2:0] Seleccion_registro_lectura=0;
    reg [2:0] Senal_de_control=0;
    reg [2:0] Inst_decodificada=0;
    reg Senal_de_salto=0;
    reg Hab=0;


    //Etapa de implemetacion
    always @(posedge i_Rst, posedge i_Timming)
    begin
        if(i_Rst) begin//Reset asincrono
            Senal_de_salto<=0;
            Selector_de_entrada_a_registros<=2'b00;
            Lectura_escritura <= 2'b00;
            Control_RX <= 3'b000;
            Control_RY <= 3'b000;
            Seleccion_registro_escritura <= 3'b000;
```

```
            Seleccion_registro_lectura <= 3'b000;
            Senal_de_control <= 3'b000;
            Inst_decodificada<=3'b000;
            Hab<=0;
        end
        else begin
            case(i_Operation_code)
            3'b000: begin
                //LOAD_1
                Senal_de_salto<=0;
                Selector_de_entrada_a_registros <= 2'b01;
                Lectura_escritura <= 2'b01;
                Control_RX <= 3'b000;
                Control_RY <= 3'b000;
                Seleccion_registro_escritura <= i_Operandos[5:3];
                Seleccion_registro_lectura <= 3'b000;
                Senal_de_control <= 3'b000;
                Inst_decodificada<=3'b000;
                Hab<=0;
            end

            3'b001: begin
                //LOAD_2
                Senal_de_salto<=0;
                Selector_de_entrada_a_registros<=2'b10;
                Lectura_escritura <= 2'b11;
                Control_RX <= 3'b000;
                Control_RY <= i_Operandos [2:0];
                Seleccion_registro_escritura <= i_Operandos [5:3];
                Seleccion_registro_lectura <= 3'b000;
                Senal_de_control <= 3'b100;
                Inst_decodificada<=3'b000;
                Hab<=0;
            end

            3'b010: begin
                //STORE_1
                Senal_de_salto<=0;
                Selector_de_entrada_a_registros<=2'b00;
                Lectura_escritura <= 2'b00;
                Control_RX <= i_Operandos[2:0];
                Control_RY <= 3'b000;
                Seleccion_registro_escritura <= 3'b000;
                Seleccion_registro_lectura <= 3'b000;
                Senal_de_control <= 3'b111;
                Inst_decodificada<=3'b000;
```

```
            Hab<=0;
        end

        3'b011: begin
            //STORE_2
            Senal_de_salto<=0;
            Selector_de_entrada_a_registros<=2'b00;
            Lectura_escritura <= 2'b00;
            Control_RX <= i_Operandos[5:3];
            Control_RY <= i_Operandos[2:0];
            Seleccion_registro_escritura <= 3'b000;
            Seleccion_registro_lectura <= 3'b000;
            Senal_de_control <= 3'b110;
            Inst_decodificada<=3'b000;
            Hab<=0;
        end

        3'b100: begin
            //MOVE
            Senal_de_salto<=0;
            Selector_de_entrada_a_registros<=2'b00;
            Lectura_escritura <= 2'b00;
            Control_RX <= 3'b000;
            Control_RY <= 3'b000;
            Seleccion_registro_escritura <= i_Operandos[5:3];
            Seleccion_registro_lectura <= i_Operandos[2:0];
            Senal_de_control <= 3'b000;
            Inst_decodificada<=3'b000;
            Hab<=0;
        end

        3'b101: begin
            //MATH
            Senal_de_salto<=0;
            Selector_de_entrada_a_registros<=2'b00;
            Lectura_escritura <= 2'b11;
            Control_RX <= i_Operandos[5:3];
            Control_RY <= 3'b000;
            Seleccion_registro_escritura <= 3'b000;
            Seleccion_registro_lectura <= 3'b000;
            Senal_de_control <= 3'b000;
            Inst_decodificada<= i_Operandos[2:0];
            Hab<=1;
        end

        3'b110: begin
```

```
                //JUMP
                case(i_Operandos[2:0])
                    3'b000: begin
                        Senal_de_salto<=1;
                        Selector_de_entrada_a_registros<=2'b00;
                        Lectura_escritura <= 2'b00;
                        Control_RX <= i_Operandos[5:3];
                        Control_RY <= 3'b000;
                        Seleccion_registro_escritura <= 3'b000;
                        Seleccion_registro_lectura <= 3'b000;
                        Senal_de_control <= 3'b000;
                        Inst_decodificada<= 3'b000;
                        Hab<=0;
                    end

                    3'b001: begin
                        Senal_de_salto<=1;
                        Selector_de_entrada_a_registros<=2'b11;
                        Lectura_escritura <= 2'b11;
                        Control_RX <= i_Operandos[5:3];
                        Control_RY <= 3'b000;
                        Seleccion_registro_escritura <= 3'b111;
                        Seleccion_registro_lectura <= 3'b000;
                        Senal_de_control <= 3'b000;
                        Inst_decodificada<= 3'b000;
                        Hab<=0;
                    end

                    3'b010: begin
                        if (i_Bandera[0]==1) begin
                            Senal_de_salto<=1;
                            Selector_de_entrada_a_registros<=2'b11;
                            Lectura_escritura <= 2'b11;
                            Control_RX <= i_Operandos [5:3];
                            Control_RY <= 3'b000;
                            Seleccion_registro_escritura <= 3'b111;
                            Seleccion_registro_lectura <= 3'b000;
                            Senal_de_control <= 3'b000;
                            Inst_decodificada<= 3'b000;
                            Hab<=0;
                        end
                    end

                    3'b011: begin
                        if (i_Bandera[0]==0) begin
                            Senal_de_salto<=1;
```

```
                    Selector_de_entrada_a_registros<=2'b11;
                    Lectura_escritura <= 2'b11;
                    Control_RX <= i_Operandos [5:3];
                    Control_RY <= 3'b000;
                    Seleccion_registro_escritura <= 3'b111;
                    Seleccion_registro_lectura <= 3'b000;
                    Senal_de_control <= 3'b000;
                    Inst_decodificada<= 3'b000;
                    Hab<=0;
                end
            end

            3'b100: begin
                if (i_Bandera[1]==1) begin
                    Senal_de_salto<=1;
                    Selector_de_entrada_a_registros<=2'b11;
                    Lectura_escritura <= 2'b11;
                    Control_RX <= i_Operandos[5:3];
                    Control_RY <= 3'b000;
                    Seleccion_registro_escritura <= 3'b111;
                    Seleccion_registro_lectura <= 3'b000;
                    Senal_de_control <= 3'b000;
                    Inst_decodificada<= 3'b000;
                    Hab<=0;
                end
            end

            3'b101: begin
                if (i_Bandera[1]==0) begin
                    Senal_de_salto<=1;
                    Selector_de_entrada_a_registros<=2'b11;
                    Lectura_escritura <= 2'b11;
                    Control_RX <= i_Operandos[5:3];
                    Control_RY <= 3'b000;
                    Seleccion_registro_escritura <= 3'b111;
                    Seleccion_registro_lectura <= 3'b000;
                    Senal_de_control <= 3'b000;
                    Inst_decodificada<= 3'b000;
                    Hab<=0;
                end
            end

            3'b110: begin
                if (i_Bandera[2]==1) begin
                    Senal_de_salto<=1;
                    Selector_de_entrada_a_registros<=2'b11;
```

```
                        Lectura_escritura <= 2'b11;
                        Control_RX <= i_Operandos[5:3];
                        Control_RY <= 3'b000;
                        Seleccion_registro_escritura <= 3'b111;
                        Seleccion_registro_lectura <= 3'b000;
                        Senal_de_control <= 3'b000;
                        Inst_decodificada<= 3'b000;
                        Hab<=0;
                    end
                end

                default: begin
                    if (i_Bandera[2]==0) begin
                        Senal_de_salto<=1;
                        Selector_de_entrada_a_registros<=2'b11;
                        Lectura_escritura <= 2'b11;
                        Control_RY <= 3'b000;
                        Seleccion_registro_escritura <= 3'b111;
                        Seleccion_registro_lectura <= 3'b000;
                        Senal_de_control <= 3'b000;
                        Inst_decodificada<= 3'b000;
                        Hab<=0;
                    end
                end
            endcase
        end

        default: begin
        //Sin operacion
            Senal_de_salto<=0;
            Selector_de_entrada_a_registros<=2'b00;
            Lectura_escritura <= 2'b00;
            Control_RY <= 3'b000;
            Seleccion_registro_escritura <= 3'b000;
            Seleccion_registro_lectura <= 3'b000;
            Senal_de_control <= 3'b000;
            Inst_decodificada<= 3'b000;
            Hab<=0;
        end
    endcase
    end
end

assign o_Senal_de_salto=Senal_de_salto;
assign o_Selector_de_entrada_a_registros=Selector_de_entrada_a_registros;
assign o_Lectura_escritura=Lectura_escritura;
```

```
    assign o_Control_RX=Control_RX;
    assign o_Control_RY=Control_RY;
    assign o_Seleccion_registro_escritura=Seleccion_registro_escritura;
    assign o_Seleccion_registro_lectura=Seleccion_registro_lectura;
    assign o_Senal_de_control=Senal_de_control;
    assign o_Inst_decodificada=Inst_decodificada;
    assign o_Hab=Hab;
endmodule
```

-Registro de instrucciones

```
//Este componente permite la decodificación y almacenamiento de la instrucció
n
//a ejecutar, la cual proviene del bus de la memoria de instrucciones, tambié
n
//permite la función de direccionamiento inmediato. Es un registro.
module Registro de instrucciones(
    input i_Timming,
    input i_Rst,
    input [8:0] i_Instrucciones,
    output [2:0] o_Instruccion,
    output [5:0] o_Operandos,
    output [7:0] o_Direccionamiento_inmediato);

    // Se declaran los registros y las señales internas
    reg [2:0] Operation_code;
    reg [5:0] Operandos;
    reg [7:0] Direccionamiento_inmediato=0;


    //Etapa de implemetacion
    always@(posedge i_Rst, posedge i_Timming)
    begin
        if (i_Rst) begin//El reset es sincrono
            Operation_code <= 3'b000;
            Operandos <= 3'b000;
        end
        else begin
            if (i_Instrucciones[8:6]==3'b000 || i_Instrucciones[8:6]==3'b010)
 begin
                Direccionamiento_inmediato<=i_Instrucciones[2:0];
                Operation_code<=i_Instrucciones[8:6];
                Operandos[5:3]<=i_Instrucciones[5:3];
                Operandos[2:0]<=& 3'b000;
            end
```

```
            else begin
                Operation_code<=i_Instrucciones[8:6];
                Operandos[5:3]<=i_Instrucciones[5:3];
                Operandos[2:0]<=i_Instrucciones[2:0];
                Direccionamiento_inmediato<=0;
            end
        end
    end

    assign o_Instruccion = Operation_code;//Se pasa el valor del registro Ope
ration
                                          //_code a o_Instruccion
    assign o_Operandos = Operandos; //Se pasa el valor del registro Operandos
 a
                                    //o_Operandos
    assign o_Direccionamiento_inmediato = Direccionamiento_inmediato;//Se pas
a el valor
                                    //del registro direccionamiento_inmediato
 a la salida
                                    //de o_Direccionamiento_inmediato
endmodule
```

-Registro de datos

```
//Este componente permite el almacenamiento de los datos provenientes de la
//memoria de datos, guarda los resultados de las operaciones realizadas por
//la ALU y guarda la direccion previa al salto del PC. Es un registro.
module Registro_de_datos(
    input i_Timming,
    input i_Rst,
    input [7:0] i_Datos,
    input [1:0] i_Lectura_escritura,
    input [2:0] i_Control_RX,
    input [2:0] i_Control_RY,
    input [2:0] i_Seleccion_registro_escritura,
    input [2:0] i_Seleccion_registro_lectura,
    output [7:0] o_RX,
    output [7:0] o_RY);

    // Se declaran los registros y las senales internas
    reg [7:0] R0;
    reg [7:0] R1;
    reg [7:0] R2;
    reg [7:0] R3;
    reg [7:0] R4;
```

```
    reg [7:0] R5;
    reg [7:0] R6;
    reg [7:0] R7;
    reg [7:0] RX;
    reg [7:0] RY;



    //Etapa de implemetacion
    always@(posedge i_Rst, posedge i_Timming)
    begin
        if (i_Rst) begin//El reset es sincrono
            R0 <= 8'b00000000;
            R1 <= 8'b00000000;
            R2 <= 8'b00000000;
            R3 <= 8'b00000000;
            R4 <= 8'b00000000;
            R5 <= 8'b00000000;
            R6 <= 8'b00000000;
            R7 <= 8'b00000000;
            RX <= 8'b00000000;
            RY <= 8'b00000000;
        end
        else begin
            case (i_Lectura_escritura)
                //Lectura de registros
                2'b00: begin
                    case (i_Control_RX)
                        3'b000: RX<=R0;
                        3'b001: RX<=R1;
                        3'b010: RX<=R2;
                        3'b011: RX<=R3;
                        3'b100: RX<=R4;
                        3'b101: RX<=R5;
                        3'b110: RX<=R6;
                        3'b111: RX<=R7;
                        default: RX<=8'b00000000;
                    endcase

                    case (i_Control_RY)
                        3'b000: RY<=R0;
                        3'b001: RY<=R1;
                        3'b010: RY<=R2;
                        3'b011: RY<=R3;
                        3'b100: RY<=R4;
                        3'b101: RY<=R5;
```

```
                    3'b110: RY<=R6;
                    3'b111: RY<=R7;
                    default: RY<=8'b00000000;
                endcase
            end

            //Escritura de registros
            2'b01: begin
                case (i_Seleccion_registro_escritura)
                    3'b000: R0<=i_Datos;
                    3'b001: R1<=i_Datos;
                    3'b010: R2<=i_Datos;
                    3'b011: R3<=i_Datos;
                    3'b100: R4<=i_Datos;
                    3'b101: R5<=i_Datos;
                    3'b110: R6<=i_Datos;
                    default: R7<=i_Datos;
                endcase
            end

            //Movimiento de registros
            2'b10: begin
                case (i_Seleccion_registro_escritura)
                    3'b000: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R0<=R0;
                            3'b001: R0<=R1;
                            3'b010: R0<=R2;
                            3'b011: R0<=R3;
                            3'b100: R0<=R4;
                            3'b101: R0<=R5;
                            3'b110: R0<=R6;
                            default: R0<=R7;
                        endcase
                    end
                    3'b001: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R1<=R0;
                            3'b001: R1<=R1;
                            3'b010: R1<=R2;
                            3'b011: R1<=R3;
                            3'b100: R1<=R4;
                            3'b101: R1<=R5;
                            3'b110: R1<=R6;
                            default: R1<=R7;
                        endcase
```

```
                    end
                    3'b010: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R2<=R0;
                            3'b001: R2<=R1;
                            3'b010: R2<=R2;
                            3'b011: R2<=R3;
                            3'b100: R2<=R4;
                            3'b101: R2<=R5;
                            3'b110: R2<=R6;
                            default: R2<=R7;
                        endcase
                    end
                    3'b011: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R3<=R0;
                            3'b001: R3<=R1;
                            3'b010: R3<=R2;
                            3'b011: R3<=R3;
                            3'b100: R3<=R4;
                            3'b101: R3<=R5;
                            3'b110: R3<=R6;
                            default: R3<=R7;
                        endcase
                    end
                    3'b100: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R4<=R0;
                            3'b001: R4<=R1;
                            3'b010: R4<=R2;
                            3'b011: R4<=R3;
                            3'b100: R4<=R4;
                            3'b101: R4<=R5;
                            3'b110: R4<=R6;
                            default: R4<=R7;
                        endcase
                    end
                    3'b101: begin
                        case (i_Seleccion_registro_lectura)
                            3'b000: R5<=R0;
                            3'b001: R5<=R1;
                            3'b010: R5<=R2;
                            3'b011: R5<=R3;
                            3'b100: R5<=R4;
                            3'b101: R5<=R5;
                            3'b110: R5<=R6;
```

```
                        default: R5<=R7;
                    endcase
                end
                3'b110: begin
                    case (i_Seleccion_registro_lectura)
                        3'b000: R6<=R0;
                        3'b001: R6<=R1;
                        3'b010: R6<=R2;
                        3'b011: R6<=R3;
                        3'b100: R6<=R4;
                        3'b101: R6<=R5;
                        3'b110: R6<=R6;
                        default: R6<=R7;
                    endcase
                end
                default: begin
                    case (i_Seleccion_registro_lectura)
                        3'b000: R7<=R0;
                        3'b001: R7<=R1;
                        3'b010: R7<=R2;
                        3'b011: R7<=R3;
                        3'b100: R7<=R4;
                        3'b101: R7<=R5;
                        3'b110: R7<=R6;
                        default: R7<=R7;
                    endcase
                end
            endcase
        end

        //Lectura y escritura simultánea
        default: begin
            case (i_Control_RX)
                3'b000: RX<=R0;
                3'b001: RX<=R1;
                3'b010: RX<=R2;
                3'b011: RX<=R3;
                3'b100: RX<=R4;
                3'b101: RX<=R5;
                3'b110: RX<=R6;
                3'b111: RX<=R7;
                default: RX<=8'b00000000;
            endcase

            case (i_Control_RY)
                3'b000: RY<=R0;
```

```
                        3'b001: RY<=R1;
                        3'b010: RY<=R2;
                        3'b011: RY<=R3;
                        3'b100: RY<=R4;
                        3'b101: RY<=R5;
                        3'b110: RY<=R6;
                        3'b111: RY<=R7;
                        default: RY<=8'b00000000;
                    endcase

                    case (i_Seleccion_registro_escritura)
                        3'b000: R0<=i_Datos;
                        3'b001: R1<=i_Datos;
                        3'b010: R2<=i_Datos;
                        3'b011: R3<=i_Datos;
                        3'b100: R4<=i_Datos;
                        3'b101: R5<=i_Datos;
                        3'b110: R6<=i_Datos;
                        default: R7<=i_Datos;
                    endcase
                end
            endcase
        end
    end

    assign o_RX = RX; //Se pasa el valor del registro RX a o_RX
    assign o_RY = RY; //Se pasa el valor del registro RY a o_RY
endmodule
```

-Unidad lógico aritmética (ALU)

```
//Este componente permite realizar operaciones del tipo lógico y matemáticas
//con los datos almacenados en registros previamente obtenidos de la memoria
//de datos, también asigna una bandera según el estado de la operación.
module ALU(
    input [2:0] i_Inst_decodificada,
    input [7:0] i_RX,
    input [7:0] i_RY,
    input i_Hab,
    output [7:0] o_Resultado,
    output [2:0] o_Bandera);

    // Se declaran los registros y las señales internas
    reg [15:0] Resultado=0;
    reg [2:0] bandera=0;
```

```
    //Etapa de implemetacion
    always@(i_Hab)
    begin
        if (i_Hab==1) begin
            case(i_Inst_decodificada)
                3'b000: Resultado <= i_RX+i_RY;
                3'b001: Resultado <= i_RX-i_RY;
                3'b010: Resultado <= i_RX<<i_RY;
                3'b011: Resultado <= i_RX>>i_RY;
                3'b100: Resultado <= ~i_RX;
                3'b101: Resultado <= i_RX & i_RY;
                3'b110: Resultado <= i_RX | i_RY;
                3'b111: Resultado <= i_RX^i_RY;
                default:
                    Resultado<=8'b00000000;
            endcase
            //Bandera zero
            if (Resultado==0) begin
                bandera[1]<=1;
            end
            else begin
                bandera[1]<=0;
            end

            //Bandera de acarreo
            if ((8'b11111111-Resultado)<0) begin
                bandera[1]<=1;
            end
            else begin
                bandera[1]<=0;
            end

            //Bandera de signo negativo
            if (i_Inst_decodificada==1 & i_RX<i_RY) begin
                bandera[0]<=1;
                end
                else begin
                bandera[0]<=0;
            end
        end
    end


    assign o_Resultado = Resultado[7:0];//Se pasa el valor del registro Resul
tado
```

```
                                        //a o_Resultado para su almacenamient
o en
                                        //los registros de datos
    assign o_Bandera =bandera; //Se pasa el valor del registro de la bandera
a la salida
                        //o_Bandera
endmodule
```

-Manager de salidas a memoria

```
//Este componente permite seleccionar el dato que entrará a los
//registros de datos. Es meramente combinacional.
module MUX
    (   input [1:0] i_Selector_de_entrada_a_registros,
        input [7:0] i_Resultado,
        input [7:0] i_Direccionamiento_inmediato,
        input [7:0] i_Bus_Datos,
        input [7:0] i_Senal_a_stack,
        output [7:0] o_Datos);

    // Se declaran los registros y las señales internas
    reg [7:0] Datos;

    //Etapa de implemetacion
    //Se incluyen todas las entradas en la lista de sensitividad,
    always@(*)
    begin
        //Dependiendo del valor del selector pasara el dato a
        //la salida
        case (i_Selector_de_entrada_a_registros)
            2'b00: Datos <= i_Resultado;
            2'b01: Datos <= i_Direccionamiento_inmediato;
            2'b10: Datos <= i_Bus_Datos;
            default : Datos <= i_Senal_a_stack;
        endcase
    end

    assign o_Datos=Datos; //Asignamos el valor del registro Datos a
                          //la salida del multiplexor

endmodule
```

-Multiplexor de entrada a registros de datos (MUX)

```
//Este componente permite seleccionar los datos que se pasarán al bus de dato
s
//de salida a memoria, al bus de direccionamiento y si se realizará un proces
o
//de lectura o escritura en memoria. Es un componente meramente combinacional
.
module Manager_de_salidas_a_memoria(
    input [7:0] i_Direccionamiento_inmediato,
    input [2:0] i_Senal_de_control,
    input [7:0] i_RX,
    input [7:0] i_RY,
    output [7:0] o_Direcciones_Datos,
    output [7:0] o_Bus_Datos,
    output o_Lectura_Escritura);

    // Se declaran los registros y las señales internas
    reg [7:0] Direcciones_Datos=0;
    reg [7:0] Bus_Datos=0;
    reg Lectura_Escritura=0;

    //Etapa de implemetacion
    //Se toman en cuenta todas las entradas para la lista de sensitividad
    always@ (*)
    begin
        if (i_Senal_de_control[2]==1) begin
            if (i_Senal_de_control[1]==0) begin
                //Lectura de memoria
                Lectura_Escritura<=0;
                Direcciones_Datos<=i_RY;
            end
            else begin
                //Escritura en memoria
                Lectura_Escritura<=1;

                if (i_Senal_de_control[0]==0) begin
                    //Direccionamiento indirecto
                    Direcciones_Datos<=i_RX;
                    Bus_Datos<=i_RY;
                end
                else begin
                    //Direccionamiento inmediato
                    Direcciones_Datos<=i_RX;
                    Bus_Datos<=i_Direccionamiento_inmediato;
                end
```

```
            end
        end
    end

    assign o_Direcciones_Datos=Direcciones_Datos;
    assign o_Bus_Datos=Bus_Datos;
    assign o_Lectura_Escritura=Lectura_Escritura;
endmodule
```

APÉNDICE B (Testbench)

-Microprocesador_Legion_H_TB

```
module Microprocesador_Legion_H_TB(
);
    reg clk,reset;
    reg [8:0] instrucciones;
    reg [7:0] bus_Datos;
    reg [31:0] frec_de_trabajo;
    wire [7:0] direcciones_Instrucciones;
    wire [7:0] direcciones_Datos;
    wire [7:0] dus_Datos;
    wire lectura_Escritura;

    Microprocesador_Legion_H DUT(
        .i_Clk(clk),
        .i_Rst(reset),
        .i_Instrucciones(instrucciones),
        .i_Bus_Datos(bus_Datos),
        .i_Frec_de_trabajo(frec_de_trabajo),
        .o_Direcciones_Instrucciones(direcciones_Instrucciones),
        .o_Direcciones_Datos(direcciones_Datos),
        .o_Bus_Datos(dus_Datos),
        .o_Lectura_Escritura(lectura_Escritura)
    );

    initial
    begin
        clk<=0;
        reset<=1;
        instrucciones<=9'b000101111;//LOAD 1,[101],111
```

```
        bus_Datos<=8'b00000011;
        frec_de_trabajo<=2;
        #20 reset<=0;
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b000100100;//LOAD 1,[100],100
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b001010100;//LOAD 2,010,[100]
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b100000010;//MOVE,000,010
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b101101000;//MATH,101,000<-SUMA
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b110101001;//JUMP 1,[101],001<-SAVE PC IN R7
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b010101111;//STORE 2,[010],111
        #5 clk<=~clk;
        #5 clk<=~clk;
    end

    always@(clk) begin
        #5 clk<=~clk;
    end

endmodule
```

-Prescalador_TB

```
module Preescalador_TB(
);
    reg clk;
    reg reset;
    reg [31:0] frec_de_trabajo;
    wire timming;

    Preescalador DUT (
        .i_Clk(clk),
        .i_Rst(reset),
        .i_Frec_de_trabajo(frec_de_trabajo),
        .o_Timming(timming)
```

```
    );

    initial begin
        clk<=0;
        reset<=1;
        frec_de_trabajo<=8;
        #20 reset<=0;
    end

    always@(clk) begin
        #5 clk <= ~clk;
    end

endmodule
```

-PC_TB

```
module PC_TB(
);
    reg clk,reset,senal_de_salto;
    reg [7:0] direccion_salto;
    wire [7:0] stack,fetch;

    PC DUT(
        .i_Timming(clk),
        .i_Rst(reset),
        .i_Senal_de_salto(senal_de_salto),
        .i_Direccion_salto(direccion_salto),
        .o_Stack(stack),
        .o_Fetch(fetch)
    );

    initial
    begin
        clk<=0;
        reset<=1;
        senal_de_salto<=0;
        direccion_salto<=0;
        #20 reset<=0;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
```

```
        #5 clk<=~clk;
        #5 clk<=~clk;
        direccion_salto<=8'b00011111;
        senal_de_salto<=~senal_de_salto;
        #5 clk<=~clk;
        #5 clk<=~clk;
        senal_de_salto<=~senal_de_salto;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;
        #5 clk<=~clk;

    end

    always@(clk) begin
        #5 clk<=~clk;
    end


endmodule
```

-Unidad_de_control_TB

```
module Unidad_de_control_TB(
);
    reg clk,reset;
    reg [2:0] bandera,operation_code;
    reg [5:0] operandos;
    wire senal_de_salto,hab;
    wire [1:0] selector_de_entrada_a_registros,lectura_escritura;
    wire [2:0] control_RX,control_RY,seleccion_registro_escritura;
    wire [2:0] seleccion_registro_lectura,senal_de_control,inst_decodificada;

    Unidad_de_control DUT(
        .i_Timming(clk),
        .i_Rst(reset),
        .i_Bandera(bandera),
        .i_Operation_code(operation_code),
        .i_Operandos(operandos),
        .o_Senal_de_salto(senal_de_salto),
        .o_Selector_de_entrada_a_registros(selector_de_entrada_a_registros),
        .o_Lectura_escritura(lectura_escritura),
        .o_Control_RX(control_RX),
        .o_Control_RY(control_RY),
```

```
        .o_Seleccion_registro_escritura(seleccion_registro_escritura),
        .o_Seleccion_registro_lectura(seleccion_registro_lectura),
        .o_Senal_de_control(senal_de_control),
        .o_Inst_decodificada(inst_decodificada),
        .o_Hab(hab)
    );

    initial
    begin
        clk<=0;
        reset<=1;
        bandera<=3'b000;
        operation_code<=3'b010;
        operandos<=6'b111101;
        #20 reset<=0;
        #5 clk<=~clk;
        operation_code<=3'b000;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b001;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b010;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b011;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b100;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b101;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        operation_code<=3'b110;
        operandos<=6'b111101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        bandera<=3'b000;
        operation_code<=3'b111;
```

```
        operandos<=6'b001101;
        #5 clk<=~clk;
        #5 clk<=~clk;
        bandera<=3'b100;
        operation_code<=3'b111;
        operandos<=6'b001110;
        #5 clk<=~clk;
        #5 clk<=~clk;
    end


endmodule
```

-Registro_de_instrucciones_TB

```
module Registro_de_instrucciones_TB(
);
    reg clk,reset;
    reg [8:0] instrucciones;
    wire [2:0] instruccion;
    wire [5:0] operandos;
    wire [7:0] direccionamiento_inmediato;

    Registro_de_instrucciones DUT(
        .i_Timming(clk),
        .i_Rst(reset),
        .i_Instrucciones(instrucciones),
        .o_Instruccion(instruccion),
        .o_Operandos(operandos),
        .o_Direccionamiento_inmediato(direccionamiento_inmediato)
    );

    initial
    begin
        clk<=0;
        reset<=1;
        instrucciones<=9'b111001110;
        #20 reset<=0;
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b011001100;
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b010001110;
        #5 clk<=~clk;
```

```
        #5 clk<=~clk;
        instrucciones<=9'b100001111;
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b000110110;
        #5 clk<=~clk;
        #5 clk<=~clk;
        instrucciones<=9'b001110110;
        #5 clk<=~clk;
        #5 clk<=~clk;
    end



endmodule
```

-Registro_de_datos_TB

```
module Registro_de_datos_TB(
);
    reg clk,reset;
    reg [7:0] datos;
    reg [1:0] lectura_escritura;
    reg [2:0] control_RX,control_RY,seleccion_registro_escritura;
    reg [2:0] seleccion_registro_lectura;
    wire [7:0] RX,RY;

    Registro_de_datos DUT(
        .i_Timming(clk),
        .i_Rst(reset),
        .i_Datos(datos),
        .i_Lectura_escritura(lectura_escritura),
        .i_Control_RX(control_RX),
        .i_Control_RY(control_RY),
        .i_Seleccion_registro_escritura(seleccion_registro_escritura),
        .i_Seleccion_registro_lectura(seleccion_registro_lectura),
        .o_RX(RX),
        .o_RY(RY)
    );

    initial
    begin
        clk<=0;
        reset<=1;
        datos<=8'b11100111;
```

```
        lectura_escritura<=2'b00;
        control_RX<=3'b001;
        control_RY<=3'b101;
        seleccion_registro_escritura<=3'b111;
        seleccion_registro_lectura<=3'b100;
        #20 reset<=0;
        #5 clk<=~clk;
        lectura_escritura<=2'b01;
        seleccion_registro_escritura<=3'b001;
        #5 clk<=~clk;
        #5 clk<=~clk;
        datos<=8'b11111111;
        seleccion_registro_escritura<=3'b101;
        lectura_escritura<=2'b01;
        #5 clk<=~clk;
        #5 clk<=~clk;
        lectura_escritura<=2'b00;
        #5 clk<=~clk;
        #5 clk<=~clk;
        lectura_escritura<=2'b10;
        seleccion_registro_lectura<=3'b001;
        seleccion_registro_escritura<=3'b111;
        #5 clk<=~clk;
        #5 clk<=~clk;
        lectura_escritura<=2'b11;
        control_RX<=3'b101;
        control_RY<=3'b111;
        datos<=8'b11111111;
        seleccion_registro_escritura<=3'b010;
    end



endmodule
```

-ALU_TB

```
module ALU_TB(
);
    reg hab;
    reg [2:0] inst_decodificada;
    reg [7:0] RX,RY;
    wire [7:0] resultado;
    wire [2:0] bandera;
```

```
    ALU DUT(
        .i_Inst_decodificada(inst_decodificada),
        .i_RX(RX),
        .i_RY(RY),
        .i_Hab(hab),
        .o_Resultado(resultado),
        .o_Bandera(bandera)
    );

    initial
    begin
        inst_decodificada<=3'b000;
        RX<=8'b00001111;
        RY<=8'b00000001;
        hab<=0;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b001;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b010;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b011;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b100;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b101;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b110;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
```

```
        #5 hab<=~hab;
        inst_decodificada<=3'b111;
        RX<=8'b00001111;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b001;
        RX<=8'b00000001;
        RY<=8'b00001111;
        #5 hab<=~hab;
        #5 hab<=~hab;
        inst_decodificada<=3'b001;
        RX<=8'b00000001;
        RY<=8'b00000001;
        #5 hab<=~hab;
        #5 hab<=~hab;
    End

endmodule
```

-Manager_de_salidas_a_memoria_TB

```
module Manager_de_salidas_a_memoria_TB(
);
    reg [7:0] direccionamiento_inmediato;
    reg [2:0] senal_de_control;
    reg [7:0] RX,RY;
    wire [7:0] direcciones_Datos;
    wire [7:0] bus_Datos;
    wire lectura_Escritura;

    Manager_de_salidas_a_memoria DUT(
        .i_Direccionamiento_inmediato(direccionamiento_inmediato),
        .i_Senal_de_control(senal_de_control),
        .i_RX(RX),
        .i_RY(RY),
        .o_Direcciones_Datos(direcciones_Datos),
        .o_Bus_Datos(bus_Datos),
        .o_Lectura_Escritura(lectura_Escritura)
    );

    initial
    begin
        direccionamiento_inmediato<=8'b11110000;
        senal_de_control<=3'b000;
```

```
        RX<=8'b11001110;
        RY<=8'b11111111;
        #5 senal_de_control<=3'b100;
        #5 senal_de_control<=3'b110;
        #5 senal_de_control<=3'b111;

    end

endmodule
```

-MUX_TB

```
module MUX_TB(
);
    reg [1:0] selector_de_entrada_a_registros;
    reg [7:0] resultado,bus_Datos;
    reg [7:0] direccionamiento_inmediato,senal_a_stack;
    wire [7:0] datos;

    MUX DUT(
        .i_Selector_de_entrada_a_registros(selector_de_entrada_a_registros),
        .i_Resultado(resultado),
        .i_Direccionamiento_inmediato(direccionamiento_inmediato),
        .i_Bus_Datos(bus_Datos),
        .i_Senal_a_stack(senal_a_stack),
        .o_Datos(datos)
    );

    initial
    begin
        selector_de_entrada_a_registros<=2'b00;
        resultado<=8'b11001110;
        direccionamiento_inmediato<=8'b00000001;
        bus_Datos<=8'b11010101;
        senal_a_stack<=8'b11111111;
        #5 selector_de_entrada_a_registros<=2'b01;
        #5 selector_de_entrada_a_registros<=2'b10;
        #5 selector_de_entrada_a_registros<=2'b11;

    End
```

```
endmodule
```