



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



# Arquitectura de computadoras

## Microprocesador de 4 bits



# LAGARTIJAX4

Prof. César Mujica Asencio - 3CM4

Equipo:

Abraham Zaid Aguilar Reyes - Brandon Iván Hernández Reséndiz -Rámses Fuentes Perez

# Índice

<b>I: Introducción</b>	1
Definiciones	2
Microporcesador	2
Arquitectura	2
Instrucciones RISC	3
Diseño	4
LagartijaX4: micro_stage 1 y 2 + GCM	4
Implementación del diseño	6
Organización del proyecto	6
Paquetes	6
Contenidos	7
Formato de archivos	7
Niveles de diseño	8
<b>II: Stage 1</b>	9
ALU + Registros ACM, ACC y RD	10
ALU 16 Operaciones	11
Diseño	11
Instrucciones	12
Unidad lógica	13
Unidad Aritmética	13
Resta	14
Bloque de control de operandos	14
Simulación	15
Registros	17
Árbol de instancias	17
Propagación de ALU	17
Simulación	18
Convenciones para la simulación	18
Testbench	20
micro_stage1	22

<b>III: Stage 2</b>	25
Manejo de signos + ROM + PC + RI + DI	26
Manejo de signos	26
Control de operación para suma/resta	27
Control de signo para suma/resta	28
Control de operación para transferencia, incremento y decremento	29
Control de signo para la transferencia, incremento y decremento	30
Simulación	31
ROM	33
Registro de instrucción RI + Decodificador de instrucción	34
Contador de programa PC	34
Instrucción de 16 bits	34
GCM	35
Descripción	36
Simulación	36
<b>IV: LagartijaX4</b>	37
Instrucciones de prueba	38
Simulación	39
Conclusión	43
<b>Bibliografía y Referencias electrónicas</b>	46
<b>Anexo A: Nomenclatura de dispositivos</b>	47



# I: Introducción

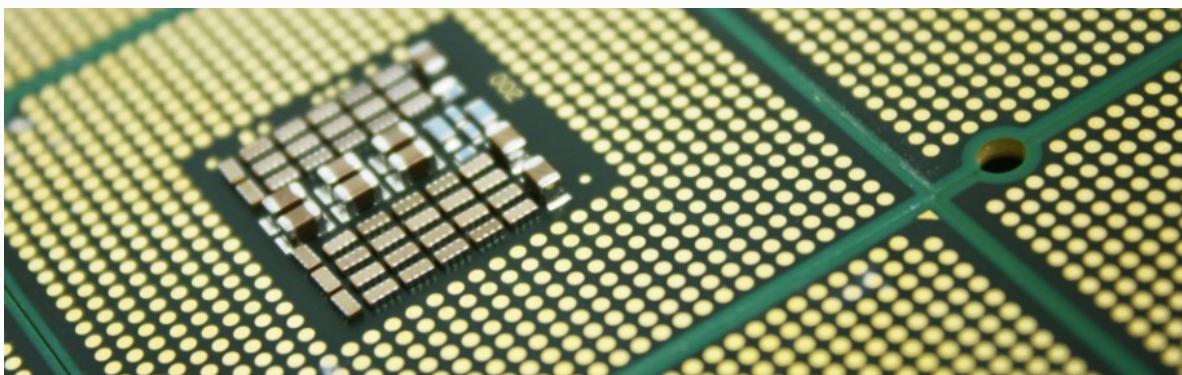
# I: Introducción

Hoy en día la familiaridad que se tiene con las tecnologías, hace que los procesos de cómputo de cualquier dispositivo electrónico sean transparentes, extremadamente rápidos y accesibles desde casi cualquier lugar en el planeta. Los dispositivos y tecnologías que se utilizan conforman una vasta variedad, sin embargo, la totalidad de estos requieren de un componente más que fundamental, el procesador o microprocesador. Como veremos a continuación en la definición de microprocesador, este junto con los diferentes mecanismos de comunicación entre dispositivos, conforman el cerebro y columna vertebral de cualquier máquina que requiera realizar cálculos. El presente trabajo describe el proceso de diseño y modelado en VHDL de un microprocesador de 4 bits de arquitectura Von Neumann con un conjunto de instrucciones basadas en el formato de instrucción RISC.

## Definiciones

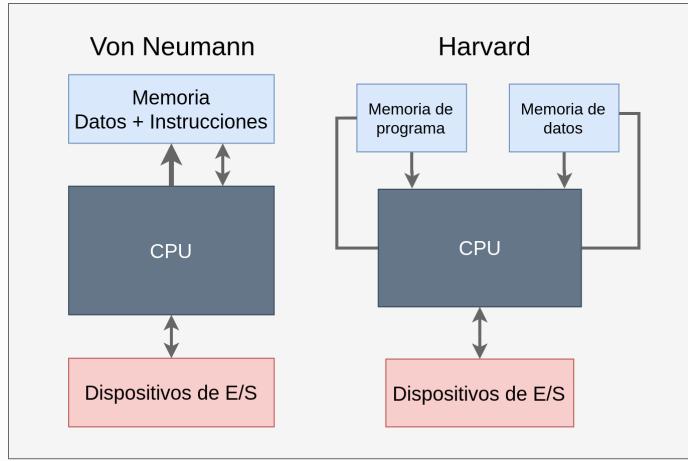
### Microprocesador

El término **microprocesador** hace referencia a un circuito integrado digital encargado de ejecutar los programas de una computadora. Dicho circuito trabaja al nivel más bajo de hardware, es decir la información que procesa se representa en forma binaria. El procesador lleva a cabo una serie de instrucciones que principalmente se refiere a la carga/escritura de valores en alguno localidad de memoria o registro y a operaciones aritméticas y lógicas sobre uno o más operandos. A pesar de que las funciones de un procesador son comunes entre los diferentes diseños y arquitecturas, estos presentan diferencias en cuanto los componentes que los conforman o la manera en que se procesa la información.



### Arquitectura

Dentro del diseño de computadoras existen dos principales tipos de arquitecturas que describen a un muy alto nivel los componentes básicos y relaciones entre estos de una computadora, la máquina Harvard y la de Von Neumann. Regularmente estas descripciones se suelen referir a la totalidad de una computadora digital, sin embargo son aplicables al concepto de microprocesador. La principal diferencia entre estas máquinas es la forma en que manejan la memoria, mientras la máquina Harvard cuenta con un par de unidades de memoria que manejan por separado los datos a procesar y las instrucciones, la máquina de Von Neumann cuenta con un solo elemento de memoria del cual se leen tanto los datos como las instrucciones.



**fig.1.0.** Descripción general de la máquina de Von Neumann y la máquina Harvard.

Si bien la diferencia entre ambas radica sólo en la forma de manejar la memoria, el desempeño total del sistema suele diferir en gran medida. En la actualidad se suele encontrar que las implementaciones basadas en la máquina Harvard son más eficientes que las implementaciones basadas en la máquina de Von Neumann, sin embargo, en épocas pasadas sucedía lo contrario, la arquitectura Von Neumann solía presentar un rendimiento mayor.

## InSTRUCCIONES RISC

Reduced Instruction Set Computer, es el significado del acrónimo RISC, el cual se refiere a un tipo de procesador que implementa en su diseño un conjunto de instrucciones reducido y de longitud fija. Una de las ventajas de utilizar un conjunto de instrucciones RISC es la rapidez en su ejecución y su sencillez, como lo menciona la siguiente cita:

[...] La principal virtud de RISC es tener un conjunto de instrucciones muy simples que se ejecutarán más rápidamente en el procesador. Existe un catálogo de pocas instrucciones y éstas son muy sencillas, lo cual implica también que para una cierta tarea compleja necesitaremos un mayor número de ellas, y por esto el programa final tendrá una longitud mayor y además accederá en un mayor número de ocasiones a los datos almacenados en la memoria. **Pablo Espeso**. CISC frente a RISC, una batalla en blanco y negro. [5]

Establecido lo anterior se hace evidente que para el diseño de un microprocesador de 4 bits, el formato de instrucciones RISC, o como lo define Behrooz [1], la filosofía RISC supone la mejor opción para la implementación de este y ademas permite una mejor descripción con fines didácticos. Como se abordará mas adelante, el conjunto de instrucciones manejados por el procesador de 4 bits, se conforma de varios campos de diferente longitud teniendo en total 16 bits por instrucción. Como se abordará mas adelante, el significado de estos campos varía de acuerdo a la operación que se realiza.

# Diseño

La implementación del microprocesador, parte de un diseño básico inicial conformado por cada uno de los componentes mínimos necesarios para el funcionamiento del microprocesador de 4 bits. Dicho diseño se segmenta en tres módulos: **Stage 1**, **Stage 2** y el Generador del ciclo de máquina **GCM**. Cada etapa en la que se subdivide la totalidad del microprocesador se considera como un módulo o componente independiente que se integraran en un componente final denominado **LagartijaX4**, esto como homenaje al procesador mexicano **Lagarto** desarrollado por el CIC.



fig.1.1. Nombre del microprocesador inspirado en el procesador Lagarto del CIC IPN

## LagartijaX4: micro\_stage 1 y 2 + GCM

La fig.1.2 muestra el diagrama a bloques del procesador de 4 bits. En este, podemos notar que todo el proceso secuencial, es orquestado por el GCM y por ende las señales, derivadas del oscilador, A-E.

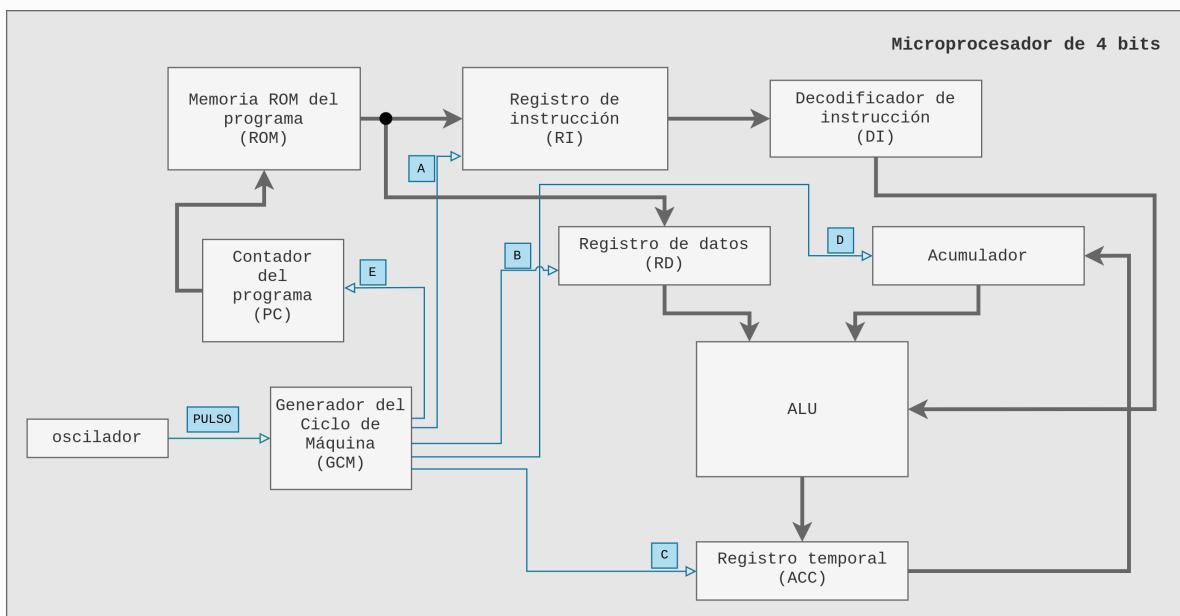
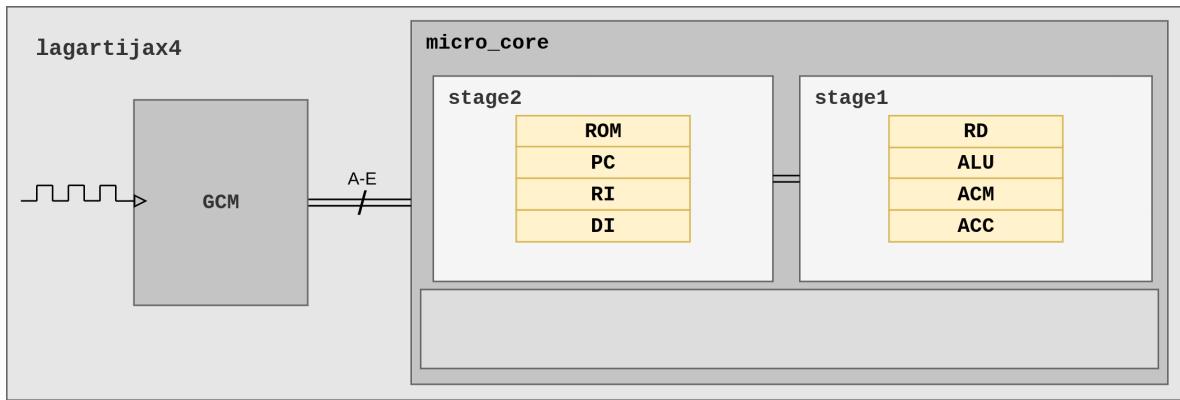


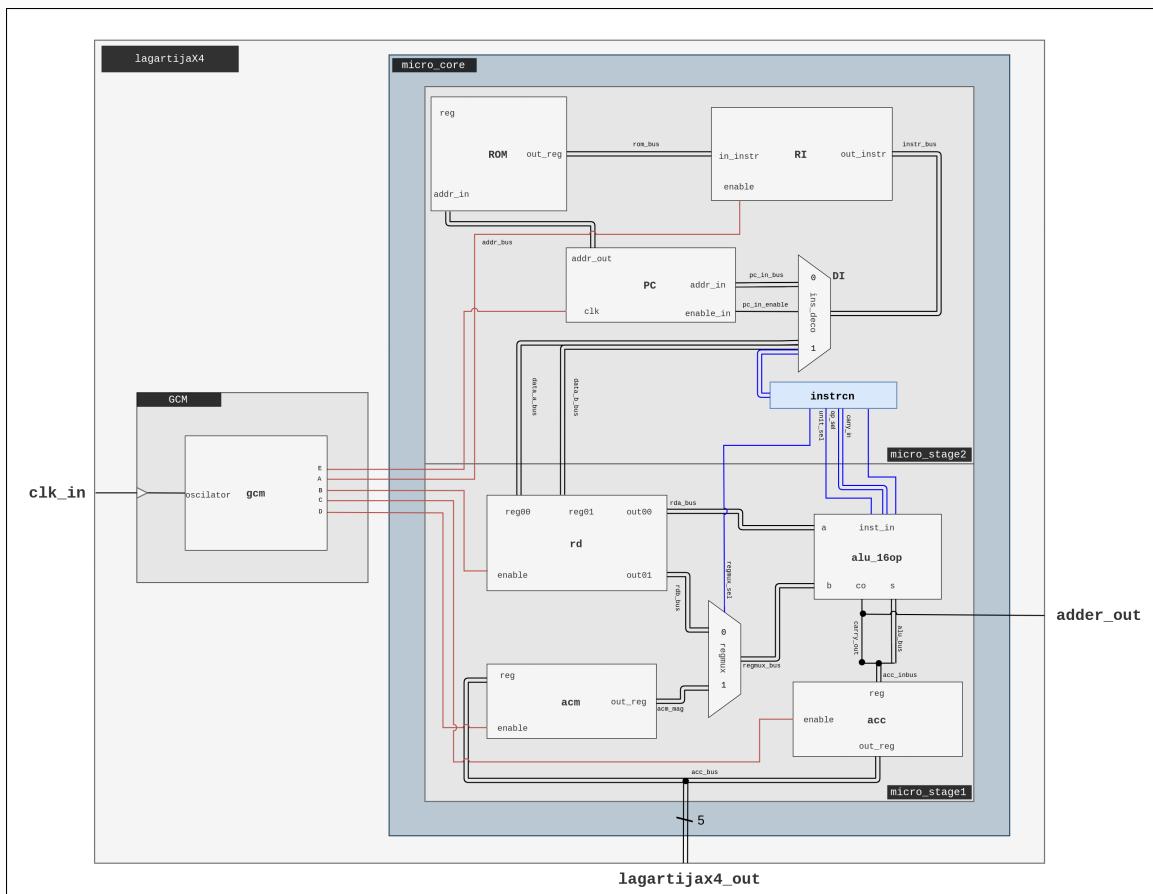
fig.1.2. Esquema general del microprocesador de 4 bits

Es por eso que los stages se dividen en función de las señales A-D y A,E para el **micro\_stage1** y **micro\_stage2** respectivamente. El siguiente diagrama muestra el diseño del procesador de manera compacta y la segmentación de este. Ambas etapas, stage 1 y 2, se agrupan a su vez en el denominado **micro\_core** que funge como contenedor y bloque receptor de las señales provenientes del GCM.



**fig.1.3.** Segmentación propuesta para el microprocesador.

La segmentación y la modulación del microprocesador permite el control y organización mas eficiente del proyecto en general; ademas de esto, facilita las tareas de adaptación o integración de nuevas funcionalidades al microprocesador, ejemplo de ello se puede observar en el Stage 2 (abordado mas adelante), donde el manejo de valores con signo es implementado sustituyendo los registro de 4 bits por registros de 5 bits para habilitar el último bit del registro como signo de la magnitud. El diagrama a bloques de la fig.1.4 representa el diseño del microprocesador tomando en cuenta las consideraciones hechas anteriormente.



**fig.1.4** Diseño del microprocesador LagartijaX4 según la segmentación propuesta y el diseño inicial dado.

# Implementación del diseño

Para la implementación del proyecto se utilizó el software de análisis y síntesis de HDL, **ISE Design Suite de Xilinx** y el simulador del lenguaje VHDL **GHDL**. El proyecto se desarrolla considerando programarse en la **FPGA Artix-7** montado en la tarjeta de desarrollo **Nexys 4** de Xilinx.

**GHDL:** Es un simulador de lenguaje VHDL de código abierto. Este permite la "compilación" (síntesis) y ejecución de código VHDL (*simulación*) en un equipo de cómputo y en conjunto con **GTKWave** la visualización del comportamiento de los modelos de manera gráfica.

## Organización del proyecto

El root del proyecto es el directorio **microprocessor**, el directorio **project** contiene el proyecto de ISE nombrado **lagartijax4**. El directorio **doc** contiene archivos de documentación, diagramas, etc. Finalmente el código VHD se organiza en el directorio **source**. Dentro de source existe un directorio llamado **sims** este contiene los archivos testbench (\*\_tb.vhd) para las simulaciones de los módulos dentro del directorio **testbench** para varias de las entidades que están declaradas en los paquetes y finalmente el directorio **gtkw** contiene los archivos de configuración de simulación de **GTKWave**.

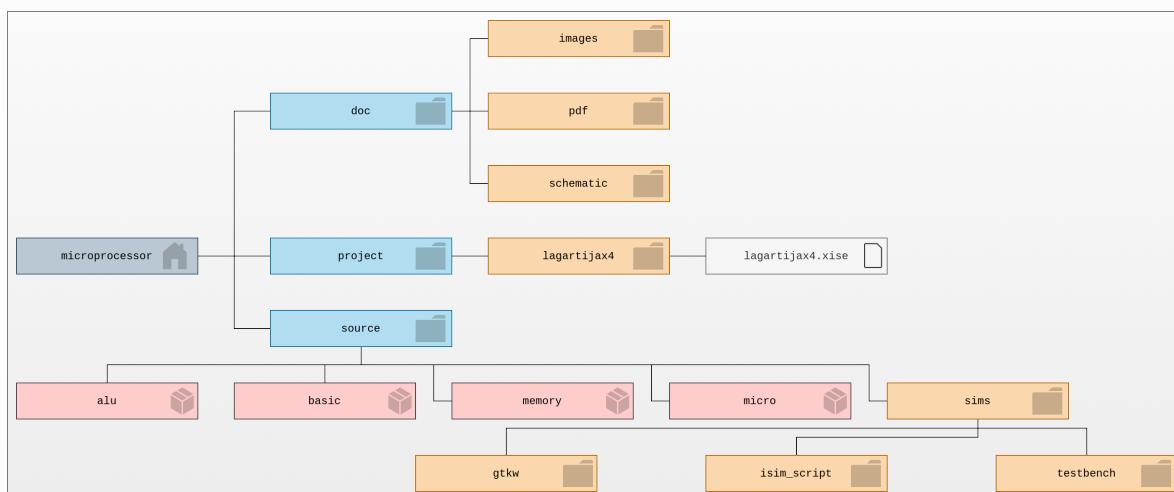


fig.1.5. Distribución del proyecto: raíz (gris), directorios (azul/naranja), directorios de paquetes (rojo).

## Paquetes

La organización de los archivos VHD del proyecto se hizo en forma de paquetes, clasificados en 4 categorías: **alu**, **basic**, **memory** y **micro**, estos paquetes se integran a la librería por defecto **work**. A cada archivo de paquete se le agrega el sufijo **\_devs** y la extensión de estos es **.vhd**. Los archivos se ubican en el directorio **source**. Dentro del proyecto de ISE se agregan los archivos fuente en forma de referencia, es decir, no son copiados al directorio de este.

En el paquete **basic** se organizan dispositivos de lógica combinacional tales como sumadores, multiplexores, comparadores, decodificadores, etc. El paquete **memory** contiene las descripciones de dispositivos de memoria, flip-flops, registros, memorias, etc. Dentro del paquete **alu** se definen los bloques que componen la ALU: la unidad lógica **logic\_u**, la unidad aritmética **arith\_u**, y demás controles. El paquete **micro** contiene la descripción de los componentes de los módulos que integran la totalidad del microprocesador: **micro\_stage1**, **micro\_stage2** y el **GMC**.

## Contenidos

La siguiente figura muestra los paquetes y los elementos declarados dentro de estos. Se considera a la librería **work** igual al directorio source del proyecto. La distribución de los directorios y carpetas en disco se hace como lo representa el árbol.

```
[alu]
alu_16op.vhd  arith_u.vhd  ctrl_arith_u.vhd  logic_u.vhd  operation_control.vhd  sign_control.vhd

[basic]
adder4bit.vhd  comp4bit.vhd  conta304.vhd  conta40F.vhd  dmux412.vhd  dmux673.vhd  logic_op_1bit.vhd  mux251.vhd
comp2bit.vhd  conta203.vhd  conta305.vhd  deco47seg.vhd  dmux513.vhd  full_adder.vhd  mux241.vhd  mux643.vhd

[memory]
bank241.vhd  bank251.vhd  bank641.vhd  ff0.vhd  reg010.vhd  reg04.vhd  reg05.vhd  rom3216.vhd

[micro]
gcm.vhd  inst_deco.vhd  micro_core.vhd  micro_stage1.vhd  micro_stage2.vhd  prog_counter.vhd
```

**fig.1.6.** Archivos por paquetes utilizados en el proyecto. La descripción de la nomenclatura utilizada para nombrar ciertos dispositivos se encuentra en el **Anexo A**.

## Formato de archivos

El contenido de los archivos de paquetes es únicamente la declaración de componentes de cada dispositivo tal cual se declara la entidad. Un archivo de paquete tiene el siguiente formato:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 package paquete is
5     component dispositivo1 is
6         port(
7             puerto1: [in|out] std_logic...;
8             puerto2: [in|out] std_logic...;
9             ...
10        );
11    end component;
12 end package;
```

La instanciación de componentes se hace mediante mapeo de puertos por posición, es decir se asignan las señales al componente en el orden en que son declarados dentro del paquete. El formato de estos archivos se asemeja al siguiente:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.paquete.dispositivo1;
4
5 entity bloque1 is
6     port(
7         puerto1_bloque: [in|out] std_logic...;
8         puerto2_bloque: [in|out] std_logic...;
9         ...
10    );
11 end entity;
12
13 architecture behavioral of bloque1 is
14     signal señal1_dispositivo: std_logic...;
15     signal señal2_dispositivo: std_logic...;
16     ...
```

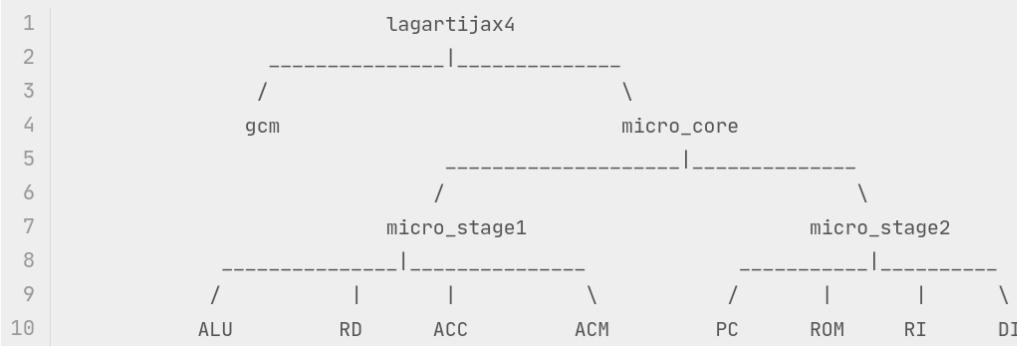
```

17 begin
18   -- Ejemplo de manipulación de señal
19   señal2_dispositivo <= not(señal1_dispositivo1) after 2 ns;
20   instancia1_dispositivo: dispositivo1 port map (señal1_dispositivo,
21   señal2_dispositivo, ...);
22   -- Ejemplo de proceso
23   process(puerto_bloque1)
24     ...
25   end if;
26   end process;
27 end architecture;

```

## Niveles de diseño

La descripción de los circuitos que integran el proyecto se hace manera estructural, por ende es posible identificar una jerarquía de niveles de diseño y de manera conveniente establecer un punto de observación del funcionamiento del módulo a diferentes escalas. Es decir, la descripción estructural nos permite situarnos en algún dispositivo en particular y observar tanto su comportamiento como el de los dispositivos en niveles inferiores.



**fig.1.7.** Niveles del diseño para el microprocesador a escala de componentes; es decir, el árbol de conforma de dispositivos integrados por dispositivos básicos.

II: Stage 1

X4

# II: Stage 1

## ALU + Registros ACM, ACC y RD

Esta primer etapa de desarrollo del microprocesador de 4 bits está compuesta por los siguientes elementos: un registro de datos RD, el registro temporal ACC, el registro acumulador ACM y una ALU de 16 operaciones de 4 bits. El propósito de este módulo del microprocesador es realizar la carga de instrucción en la ALU, la carga de los datos desde memoria en el registro RD y finalmente la carga del resultado de la operación (salida de la ALU) tanto en el registro ACC como en el ACM. Siguiendo el esquema general del microprocesador (fig.1.2) observamos que las acciones antes mencionadas se realizan en sincronía con las 4 primeras señales del GCM del microprocesador.

Los registros utilizados consisten en un arreglo de flip flops (4 o 5 dependiendo el tamaño de la palabra a almacenar) tipo D. Dado que el registro RD debe almacenar dos datos leídos desde memoria, se utiliza un banco de registros de 2x4 (elementos x tamaño de elementos en bits). Una de las salidas de dicho banco va directamente a la ALU y la otra entra en un multiplexor de 2 a 1, en el cual se hace la selección del segundo dato a operar, entre el almacenado en el registro B del banco RD y el almacenado en el registro ACM. Una vez en la ALU se realiza la operación entre los datos y el resultado, tanto magnitud (4 bits) como acarreo/signo (1 bit), se cargan en el registro ACC y en el ACM, ambos, registros de 5 bits.

El despliegue de la información se hace en un arreglo de 5 displays de 7 segmentos los cuales muestran: dato a, dato b, instrucción actual, acarreo/signo y resultado (magnitud). Para ello se implementa un decodificador de 4 bits para cada de display de 7 segmentos. Las primeras tres señales de los primeros tres displays, se toman directamente del bus de entrada al módulo, y tanto el acarreo como el valor de salida de la operación se toman desde las salidas del módulo. El diagrama a bloques correspondiente se muestra a continuación.

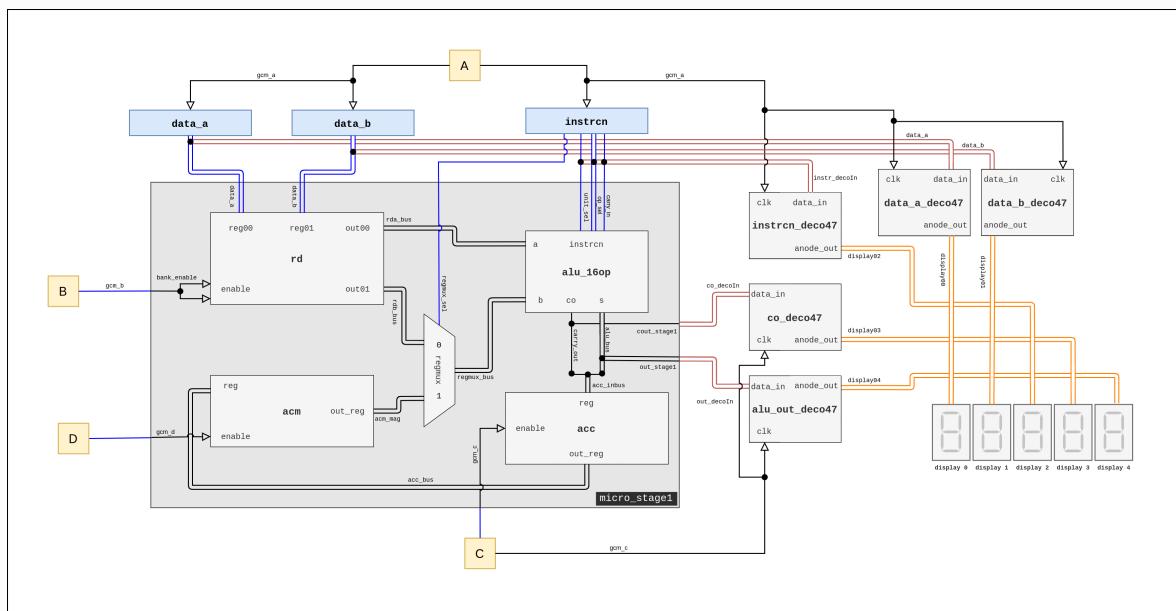


fig.2.1. Diagrama Stage1: ALU + Registros ACM, ACC y RD

El diagrama es una representación del nivel mas alto del módulo, en este se denotan los bloques, señales y buses que lo conforman. Se debe tener en consideración que los elementos externos al micro\_stage1 son los necesarios para montar un testbench, es decir, no son parte de la estructura final del módulo, su función es simplemente emular los demás dispositivos que dentro del microprocesador y proveer un medio de visualización del funcionamiento de esta primera etapa.

## ALU 16 Operaciones

Recordando la definición de microprocesador, se refiere a este como el "cerebro" de una computadora, pues es el encargado de ejecutar los programas del sistema operativo así como los programas de usuario; esta ejecución implica principalmente un conjunto de operaciones básicas en forma binaria, tarea de la cual la Unidad Aritmética Lógica (ALU) es la encargada.

### Diseño

La unidad de procesamiento se divide en dos sub-unidades: la lógica y la aritmética (Fig.1). Estas dos unidades se han considerado como dos bloques independientes `arith_u` y `logic_u`. Cada sub-unidad es capaz de realizar 8 operaciones cada una, sobre uno o los dos operandos que se especifiquen de entrada. La selección de la unidad y de la operación se hace en función de los valores de los campos recibidos en la instrucción. Esta instrucción es un conjunto de bits que se reciben desde el codificador de instrucción, elemento de una segunda etapa del microprocesador, por lo cual para pruebas y depuración de este módulo se sustituye dicho conjunto de bits provenientes del decodificador por un vector de 5 bits `instrcn` (Tabla 2.1).

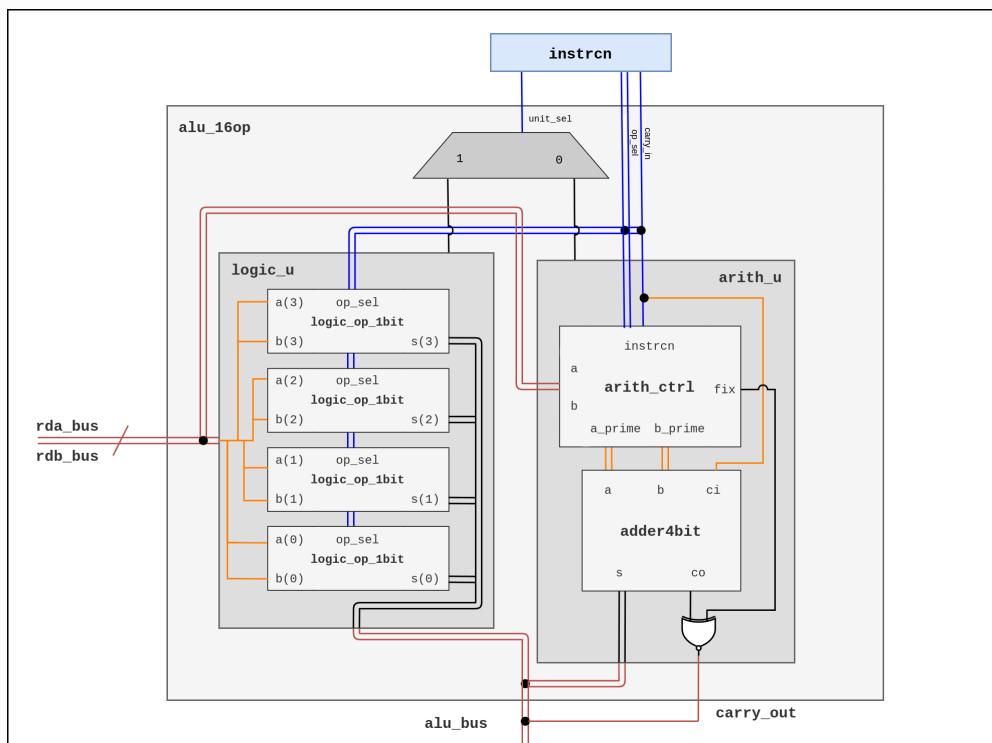


fig.2.2 Diagrama de bloques de la ALU 16 operaciones.

## Instrucciones

El formato de las instrucciones de la ALU es una palabra de 5 bits la cual se divide en campos como se muestra a continuación:

1	bit : 4	3	2-1	0
2	señal : [regmux_sel]	[unit_sel]	[op_sel]	[carry_in]

- **regmux\_sel:** El selector de registro del segundo operando se conecta al multiplexor y mediante este se elige el segundo operando que entra a la ALU, entre el dato de entrada B (rdb\_bus) y el valor almacenado en el acumulador (acm\_bus).
- **unit\_sel:** El selector de unidad determina si se realiza una operación aritmética (0) o lógica (1) sobre los operandos.
- **op\_sel & carry\_in:** La instrucción de operación concatenada con el acarreo de entrada ci determina cual de las 8 operaciones en cada una de las unidades se va a realizar.

Tabla de instrucciones

unit_sel	op_sel & ci	hex	operación
0	000	0	suma
0	001	1	resta
0	010	2	transferencia A
0	011	3	incremento A
0	100	4	transferencia B
0	101	5	incremento B
0	110	6	decremento A
0	111	7	decremento B
1	000	8	AND
1	001	9	NAND
1	010	A	OR
1	011	B	NOR
1	100	C	XOR
1	101	D	XNOR
1	110	E	BUFFER A
1	111	F	NOT A

Tabla 2.1. Código de instrucciones ALU 16 bits

## Unidad lógica

La unidad lógica se compone de 4 bloques de operadores lógicos de 1 bit (como se observa en la fig.2.2), cada uno de estos operadores realiza la operación bit a bit de las entradas. El comportamiento de cada uno de estos 4 bloques está dado por las ecuaciones de la Tabla 2.2, la cuales en función del valor de op\_sel (bits 2-1 de la instrucción) definen la salida del bloque dados los valores del bit correspondiente al operando A,B y el acarreo ci; el acarreo de salida en esta unidad se establece en 0.

op_sel	s
00	$ci \oplus (ab)$
01	$ci \oplus (a + b)$
10	$ci \oplus (a \oplus b)$
11	$ci \oplus a$

Tabla 2.2. Ecuaciones de salida por bloque lógico en función del campo op\_sel.

## Unidad Aritmética

La unidad aritmética de la ALU se compone de un bloque de control en el que en función de la operación recibida como instrucción se modifican los datos de entrada para posteriormente enviarse a un sumador completo de 4 bits. La tarea del bloque de control es modificar el valor de los operandos para obtener la operación especificada, como se muestra en la fig2.2. estos valores se ingresan en el sumador. Por ejemplo: Si se trata de una operación de incremento, el valor del segundo operando se establece a 0 y se suma el acarreo de entrada correspondiente al bit de control de la operación. La Tabla 2.3 muestra esta relación entre operando de salida con el código de operación, donde **a** y **b** son los datos de entrada y **a\_prime** y **b\_prime** los datos de salida. Un caso particular se da con la operación de resta, esta se describe con mas de detalle posteriormente.

op_sel	ci	operación	a_prime	b_prime	fix
00	0	suma	a	b	0
01	0	transferencia A	a	0000	0
10	0	transferencia B	0000	b	0
11	0	decremento A	a	1111	1
00	1	resta	b	not a	/
01	1	incremento A	a	0000	0
10	1	incremento B	0000	b	0
11	1	decremento B	1110	b	1

Tabla 2.3. Relación entre código de operación y valor de salida de los operandos. El valor de fix en la fila 5 se especifica en la tabla 2.4.

En la última columna de la tabla 2.3 se observa el valor de la bandera **fix**. Esta bandera indica si el valor del acarreo de salida debe ser negado para ser coherente con la operación realizada. El valor de **fix** entra junto con el acarreo de salida del sumador a una compuerta XNOR para obtener el valor correcto del acarreo de salida. Por ejemplo: Si el operando  $a=0011=3$  y  $b=0111=7$ , el resultado esperado de la operación  $B-A$  es 4. Sin embargo, dado el algoritmo de resta como suma con complemento a dos, el valor de  $B-A$  es -4, para obtener a la salida de la ALU el valor correcto se debe encender la bandera **fix**. El siguiente diagrama muestra el ejemplo mencionado anteriormente.

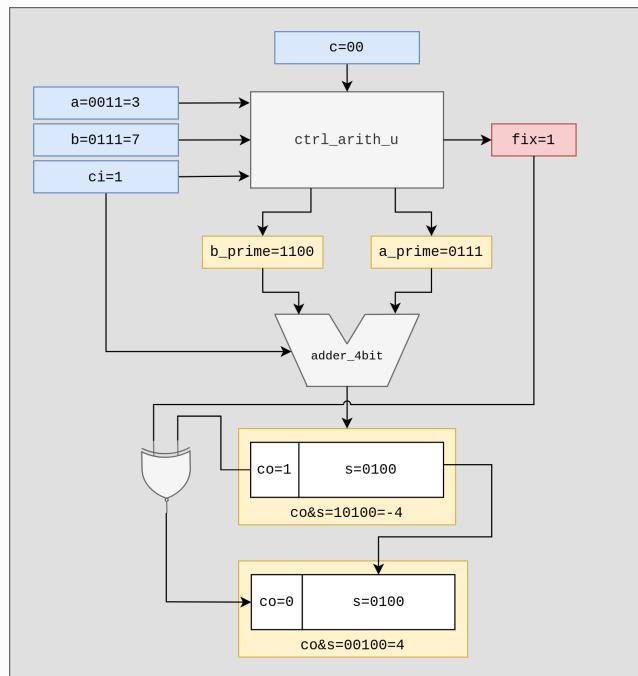


fig.2.3. Ejemplo de operación resta cuando  $\text{fix}=1$ .

## Resta

Se considera el orden de los operandos como  $B - A$ , lo cual implica que:

1. Si  $A > B$  se niega el operando B y la bandera **fix** permanece en bajo.
2. Si  $A \leq B$  el operando B se iguala al operando A negado y la bandera **fix** pasa a alto.

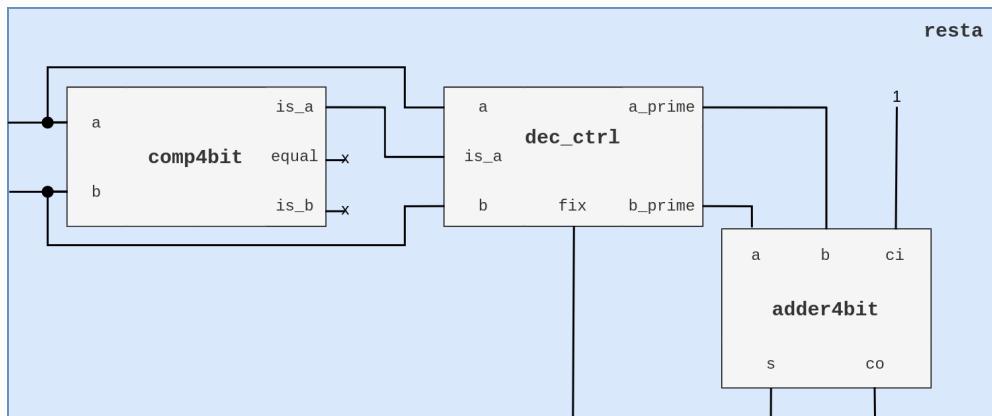
Condición	Signo
$A > B$	-
$A \leq B$	+

## Bloque de control de operandos

Para llevar a cabo el control de la bandera **fix** y el orden de los operandos adecuado para efectuar la resta, se emplea un bloque del cual su comportamiento lo describe la siguiente tabla, donde el valor **is\_a** indica con valor en alto que el operando A es mayor a B. La figura correspondiente a la resta, muestra una aproximación por bloques de la operación.

is_a	A	B	A'	B'	fix
0	0	0	$B$	$\bar{A}$	1
0	0	1	$B$	$\bar{A}$	1
0	1	0	$B$	$\bar{A}$	1
0	1	1	$B$	$\bar{A}$	1
1	0	0	$A$	$\bar{B}$	0
1	0	1	$A$	$\bar{B}$	0
1	1	0	$A$	$\bar{B}$	0
1	1	1	$A$	$\bar{B}$	0

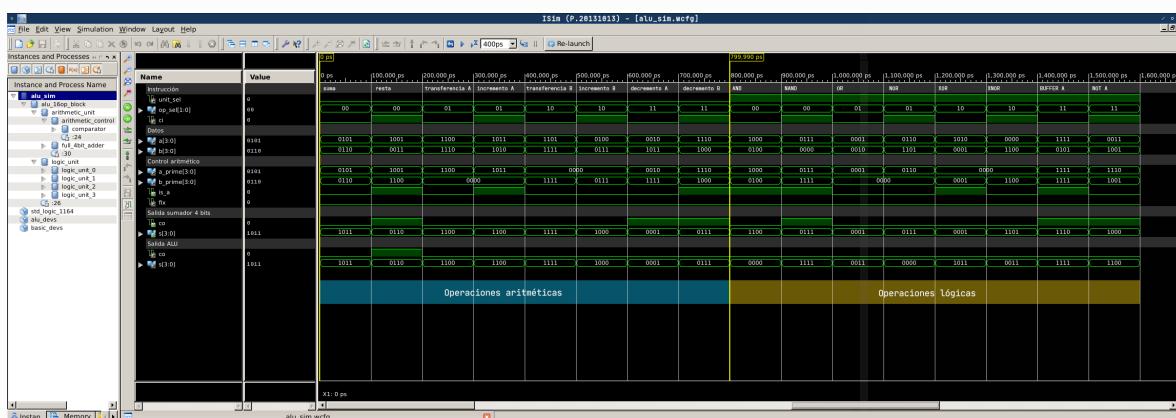
**Tabla 2.4.** Comportamiento de los operandos de salida del control de la unidad aritmética en función del valor del operando A.



**fig.2.4.** Representación como diagrama de bloques de la operación resta.

## Simulación

Se monta un entorno de simulación para observar los resultados de la ALU según la operación. La simulación se lleva a cabo en el simulador ISim de ISE, dentro de la cual se definen (forzar constante) los valores tanto de los operandos A y B como de la instrucción.



**fig.2.5.** Simulación en ISim del funcionamiento de la ALU con valores de A y B aleatorios.

Los valores forzados para la simulación (instrcn, A y B) y los resultados esperados a la salida de la ALU (co y s) se muestran en la Tabla 2.5. La fig.2.6. muestra los resultados de la simulación tanto para la unidad aritmética como para la unidad lógica.

instrcn	A	B	co	s
0000	0101	0110	0	1011
0001	1001	0011	1	0110
0010	1100	1110	0	1100
0011	1011	1010	0	1100
0100	1101	1111	0	1111
0101	0100	0111	0	1000
0110	0010	1011	0	0001
0111	1110	1000	0	0111
1000	1000	0100	0	0000
1001	0111	0000	0	1111
1010	0001	0010	0	0011
1011	0110	1101	0	0000
1100	1010	0001	0	1011
1101	0000	1100	0	0011
1110	1111	0101	0	1111
1111	0011	1001	0	1100

Tabla 2.5. Entradas y salidas esperadas en la simulación.

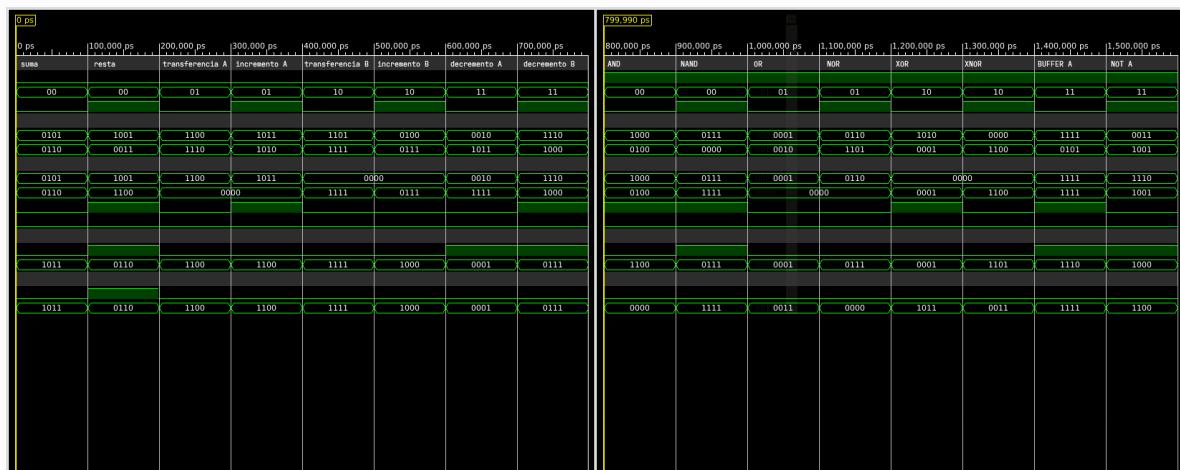


fig.2.6. Simulación de las operaciones aritméticas (a) y lógicas (b). De arriba hacia abajo: unit\_sel, op\_sel, ci,a,b,a\_prime, b\_prime, is\_a\_fix, co y s (sumador), co y s (ALU).

# Registros

Cada uno de los registros utilizados en este módulo cuenta con un puerto de activación **enable**, un bus de entrada y otro de salida de 4 ó 5 bits. El banco de registros es un arreglo de n registros, y la activación de estos se hace mediante una señal de  $2^n$  bits, donde el bit 0 corresponde a la señal enable del registro 0, el bit 1 corresponde a la señal enable del registro 1 y de manera sucesiva hasta el bit n-1.

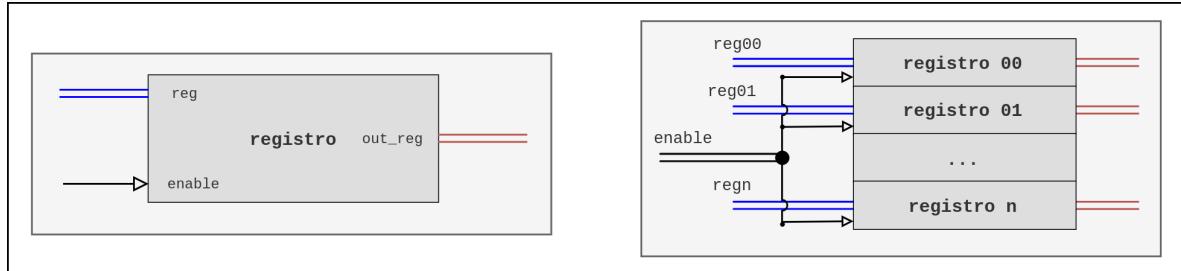


fig.2.7 Diagrama de registros y banco de registros.

## Árbol de instancias

Representa de forma jerárquica los niveles de diseño y denota las instancias que se hacen en cada componente que integra el módulo. Cada hoja se compone del nombre de la instancia seguido de su tipo. Como se observa el dispositivo con el mayor número de instancias y niveles inferiores es la **alu**. Para efectos de esta etapa, se define el módulo **micro\_stage1** como el nivel superior inicial y a partir de este se derivan los niveles subsecuentes haciendo uso de una estructura de árbol.

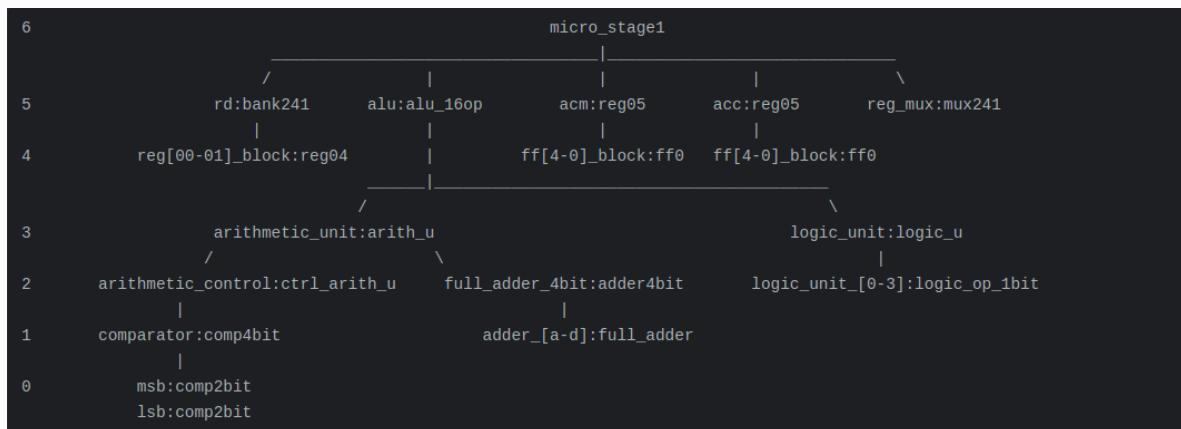


fig.2.8. Árbol de instancias para el stage 1.

## Propagación de ALU

Al ser el sumador de 4 bits el componente fundamental de la unidad aritmética, la propagación de la señal de salida y por ende el resultado de operación atraviesa diferentes dispositivos, la siguiente tabla muestra dicha ruta de abajo a arriba.

Dispositivo	Instanciado en	Señal de acarreo	Señal de salida
full_adder	adder4bit	co	s
adder4bit	arith_u	co	s
arith_u	alu_16op	arith_unit_coutput	arith_unit_output
alu_16op	micro_stage1	carry_out	alu_bus

Tabla 2.6. Propagación de ALU

## Simulación

### Niveles de simulación

Para llevar a cabo este proceso el proyecto se carga al software [ISE Design Suite](#) y se corren las respectivas simulaciones en [ISim](#). Para observar a detalle el comportamiento del módulo se realiza la simulación en dos niveles:

- 1. Testbench:** Definimos un nivel extra a los niveles mostrados en el árbol de instancias. Este nivel simula un componente de caja cerrada, es decir, un componente sin puertos de entrada ni de salida, en el cual se crea una instancia del siguiente nivel, [micro\\_stage1](#), y las señales provenientes de los dispositivos con los que este interactúa dentro del microprocesador son emuladas en forma de valores constantes. Se implementa un contador de 0 a 3 conectado a un demultiplexor para emular el funcionamiento del GCM. Para el despliegue de los resultados se hacen las instancias de los 5 decodificadores de 4 bits a displays de 7 segmentos que se muestran en el diagrama del módulo. El testbench se describe en el archivo [vhdl stage1\\_sim.vhd](#).
- 2. micro\_stage1:** Dentro de la simulación a este nivel se presta especial atención a las 4 acciones principales de las que se encarga este módulo: la carga de instrucción en la ALU, la carga de datos en el registro RD, la carga de la salida de la ALU en el registro ACC y el envío de este valor al ACM.

### Convenciones para la simulación

#### Frecuencia y periodo

Se supone una frecuencia de reloj de 2.5 GHz, con lo cual determinamos el periodo correspondiente y el tiempo total necesario por el módulo para realizar una instrucción:

$$T_{clk} = \frac{1}{f_{clk}} = \frac{1}{2.5 \times 10^9 Hz} = 4 \times 10^{-10} s = 400 ps$$

$$t_{stage1} = 4T_{clk} = 4 \cdot 400 ps = 1600 ps = 1.6 ns$$

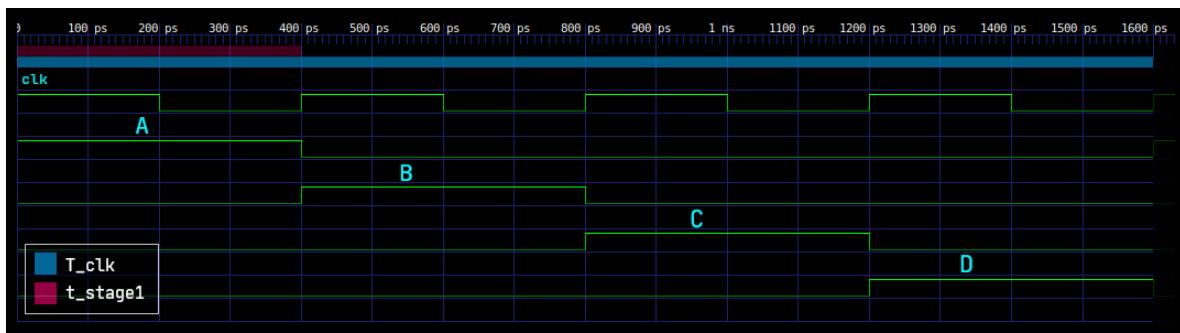


fig.2.9. Señales A-D para efectos de simulación, correspondientes a las generadas por el GCM.

### Display de 7 Segmentos

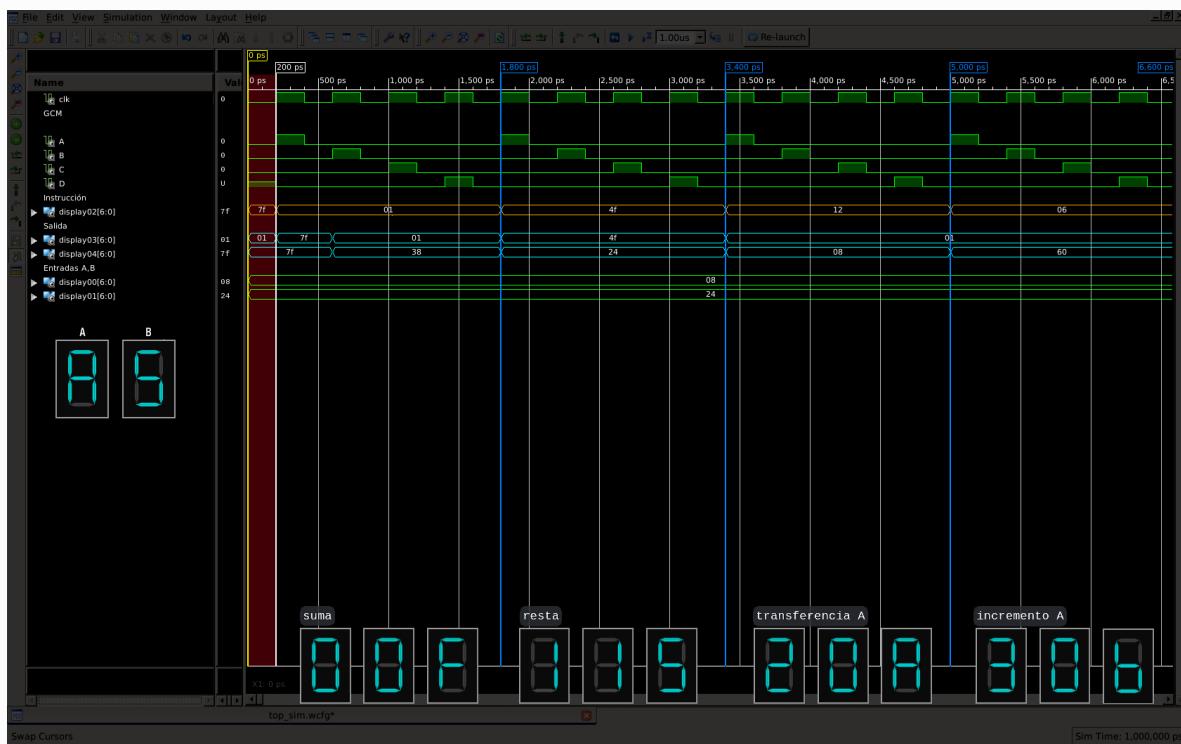
Como se ha indicado anteriormente, la simulación del testbench obedece al diagrama del modulo mostrado en la fig.1 y por ende el despliegue de la información se da a través de displays de 7 segmentos. Para interpretar de forma mas ágil los resultados en la simulación se utiliza la siguiente tabla con la representación en binario, hexadecimal y gráfica de las posibles salidas del decodificador de 4 bits a 7 segmentos.

bin	hex	gráfico	bin	hex	gráfico	bin	hex	gráfico
111111	7F		01000100	24		11000000	60	
00000001	01		01000000	20		11100110	72	
1001111	4F		0001111	0F		10000110	42	
00100010	12		00000000	00		01100000	30	
0000110	06		00000100	04		01110000	38	
10011100	4C		0001000	08				

Tabla 2.7. Display de 7 segmentos.

## Testbench

Las figuras (2.10-2.17) que se presentan a continuación corresponden a la simulación del testbench, en este se realiza la carga de cada una de las 16 operaciones que realiza la ALU con dos operandos constantes  $A = 1010$  y  $B = 0101$ . Como se definió anteriormente, el tiempo necesario para realizar los 4 procesos habilitados por las señales A-D es de  $1.6 \text{ ns} = 1600 \text{ ps}$ , periodo que se denota en la simulación mediante marcadores. La señal correspondiente al bus de entrada de instrucción a la ALU y al display 2 se resalta con color anaranjado y las señales correspondientes a los buses que conectan los displays 3 y 4, acarreo y salida respectivamente, se resaltan con color azul. La parte inferior muestra la representación gráfica de los displays 2,3 y 4 en función del valor de las señales antes mencionadas. Por encima de estos displays se indica la operación que se realiza en ese periodo de tiempo.



**fig.2.10. Operaciones [0-3].** Se denota con color rojo el retraso de  $200 \text{ ps}$  que se obtiene debido a el inicio del reloj en 0. Para el caso de la primera operación el valor de los displays 3 y 4 en los primeros  $600 \text{ ps}$  es igual a  $4F$  que representa el display apagado. Esto debido a la secuencia de operaciones que realiza el módulo en sincronía con las señales A-D. Como se observa en el diagrama del módulo, la señal de acarreo y la señal de salida de la ALU toman el valor del resultado de la operación hasta que se cargan los registros en la ALU con la señal B, por ende, valor de los displays se actualiza con cada pulso alto de la señal B.



fig.2.11. Operaciones [4-7].



fig.2.12. Operaciones [8-11].



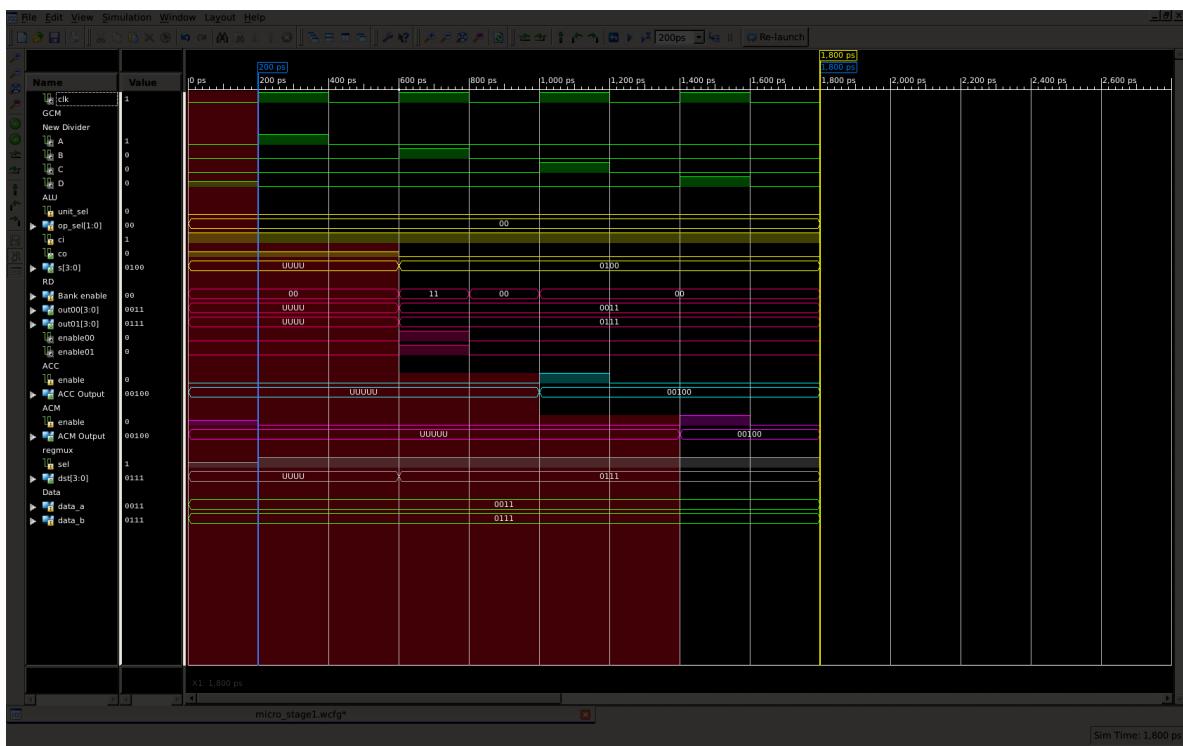
fig2.13. Operaciones [C-F].

## micro\_stage1

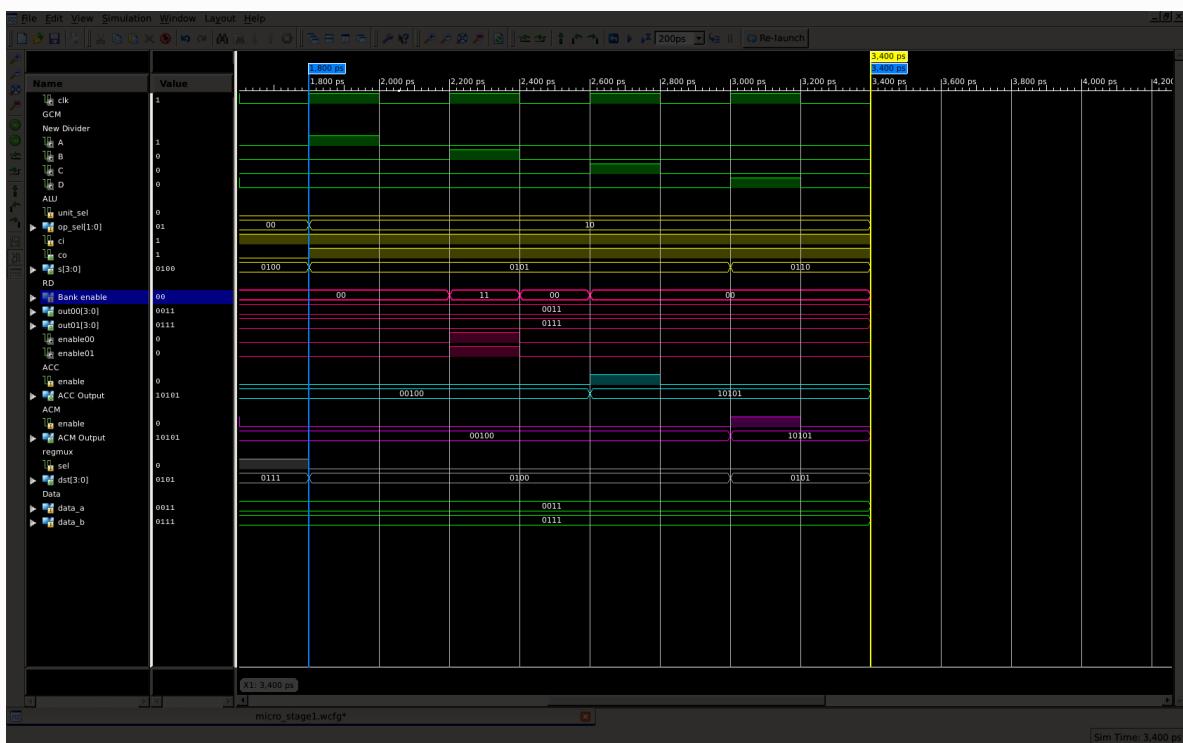
En este nivel de simulación se fuerzan las constantes  $A = 0011$  y  $B = 0111$  para los valores de los buses *data\_a* y *data\_b* respectivamente y la siguiente secuencia de instrucciones:

Operación	Instrucción	Resultado
B-A	10001	0111-0011 = 0100
ACM++	00101	0100+0001 = 0101
A and ACM	01000	0011 and 0101 = 0001
ACM--	00111	0001-0001 = 0000

Las señales de entrada de la ALU correspondientes a la instrucción *unit\_sel*, *op\_sel*, *ci*, el acarreo de salida *co* y el valor de salida 4 bits *s* se identifican con color amarillo. La señal de enable y el bus de salida del banco de registro RD, se identifican con color rosa. Las señales del registro ACC se identifican con color cyan y el registro ACM color fucsia. Las señales del multiplexor regmux se identifican con color gris y finalmente los valores de entrada al módulo con color verde.



**fig.2.14. B-A.** En la ejecución de la primer instrucción durante el pulso A se carga la instrucción a la ALU y se hace la selección de B como segundo operando. Pasados  $600\text{ ps}$  durante el pulso B se activa el banco de registros RD y se obtiene la salida de la operación de la ALU en el bus correspondiente. Para el pulso C,  $1000\text{ ps}$  después, se carga el resultado de la operación en el registro ACC y finalmente pasados  $1200\text{ ps}$  se carga el valor de ACC en ACM. Este proceso se repite a partir de cada cambio de instrucción o pulso positivo de la señal A. El tiempo inactivo o de valor indefinido de cada dispositivo se denota con un sombreado rojo en la figura. Estos tiempos inactivos se deben a la secuencia de las señales A-C.



**fig.2.15. ACM++.** Una vez ejecutada la primer instrucción los valores de los registro se actualizan cada pulso del GCM correspondiente.

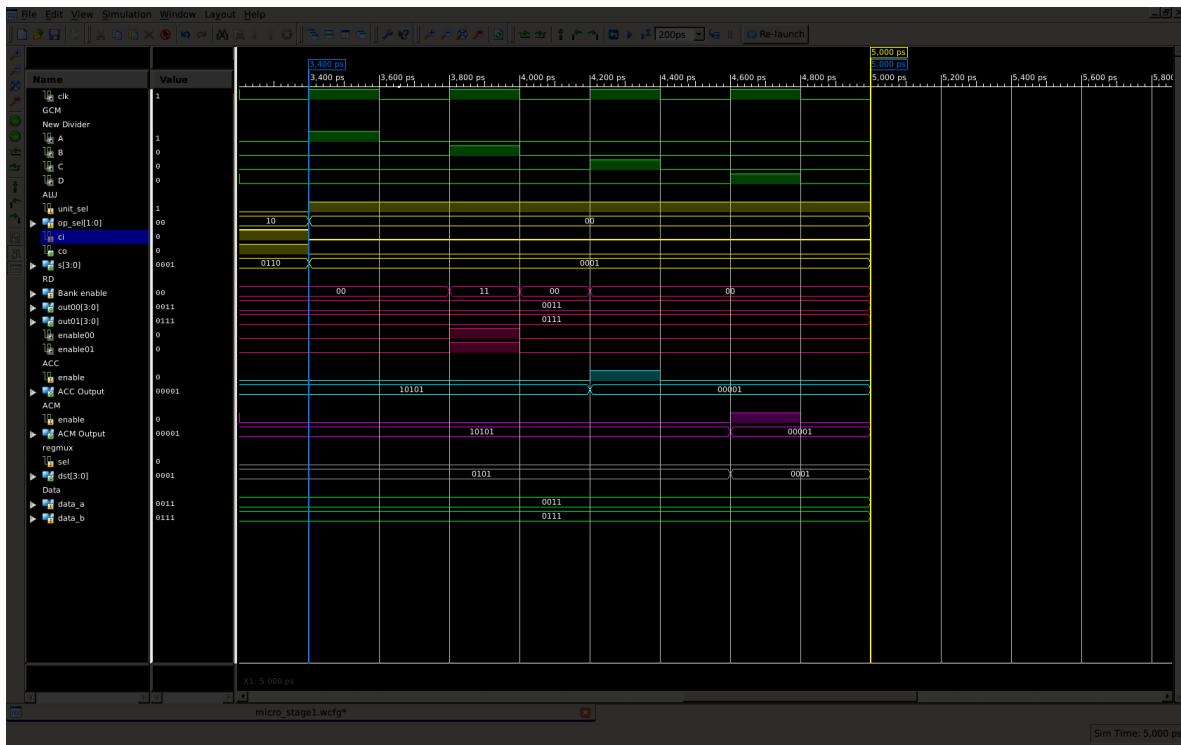


fig.2.16. A and ACM.

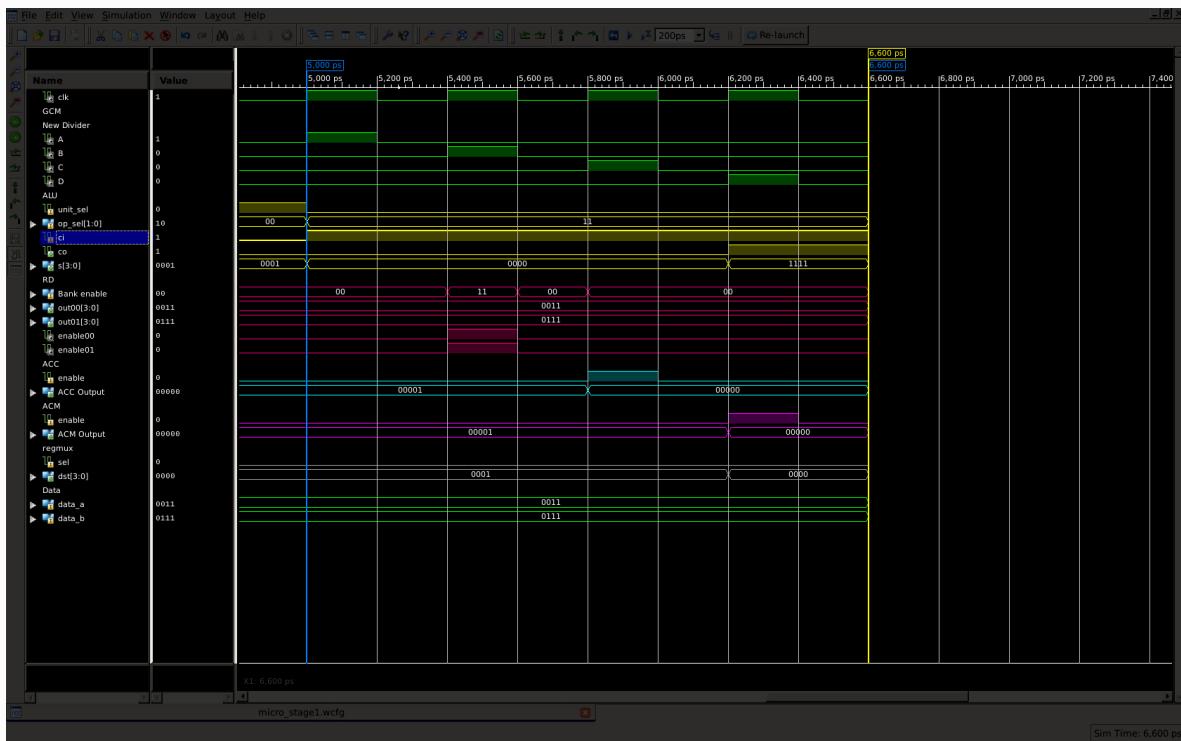


fig.2.17.ACM--.

## III: Stage 2

X4

# III: Stage 2

## Manejo de signos + ROM + PC + RI + DI

En esta segunda etapa del microprocesador, ademas de realizar la extensión de los registros del Stage 1 a 5 bits, así como la integración de un conjunto de bloques adicionales a la ALU para permitir el manejo de valores de 4 bits con signo, se implementa el segundo módulo que corresponde a la unidad de control. La siguiente figura muestra el diagrama correspondiente a esta etapa.

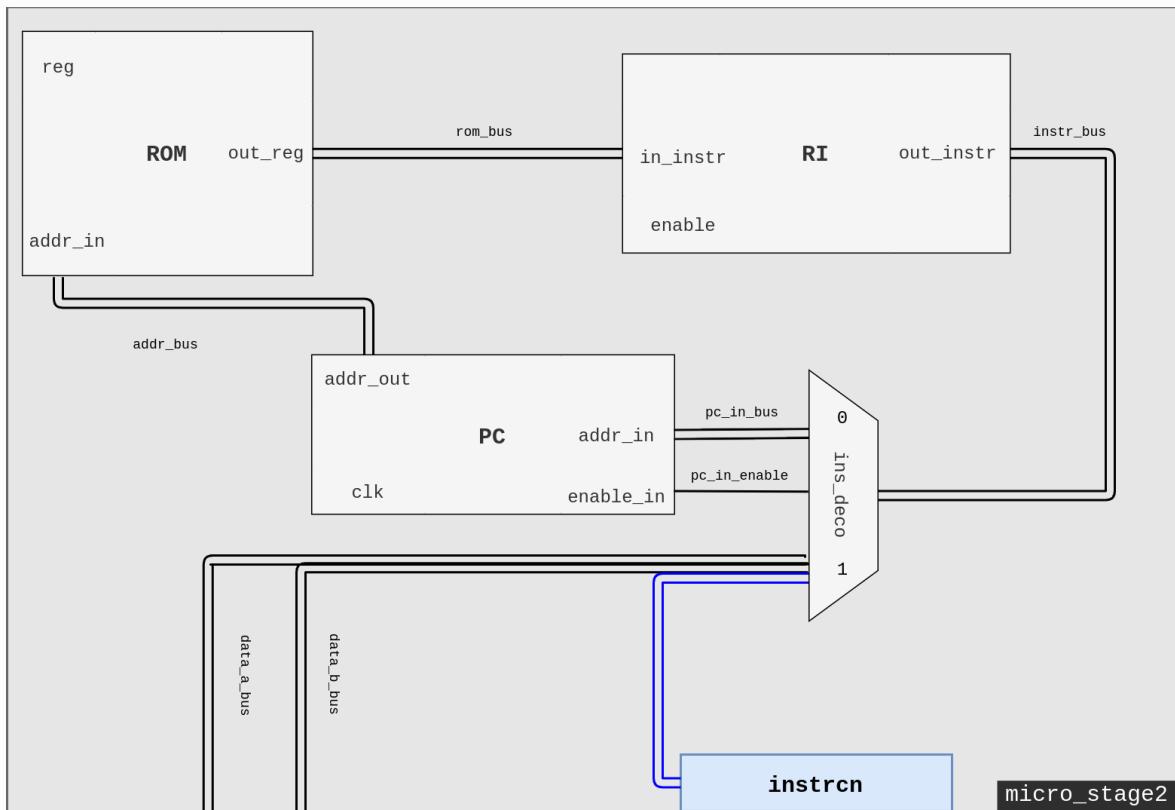
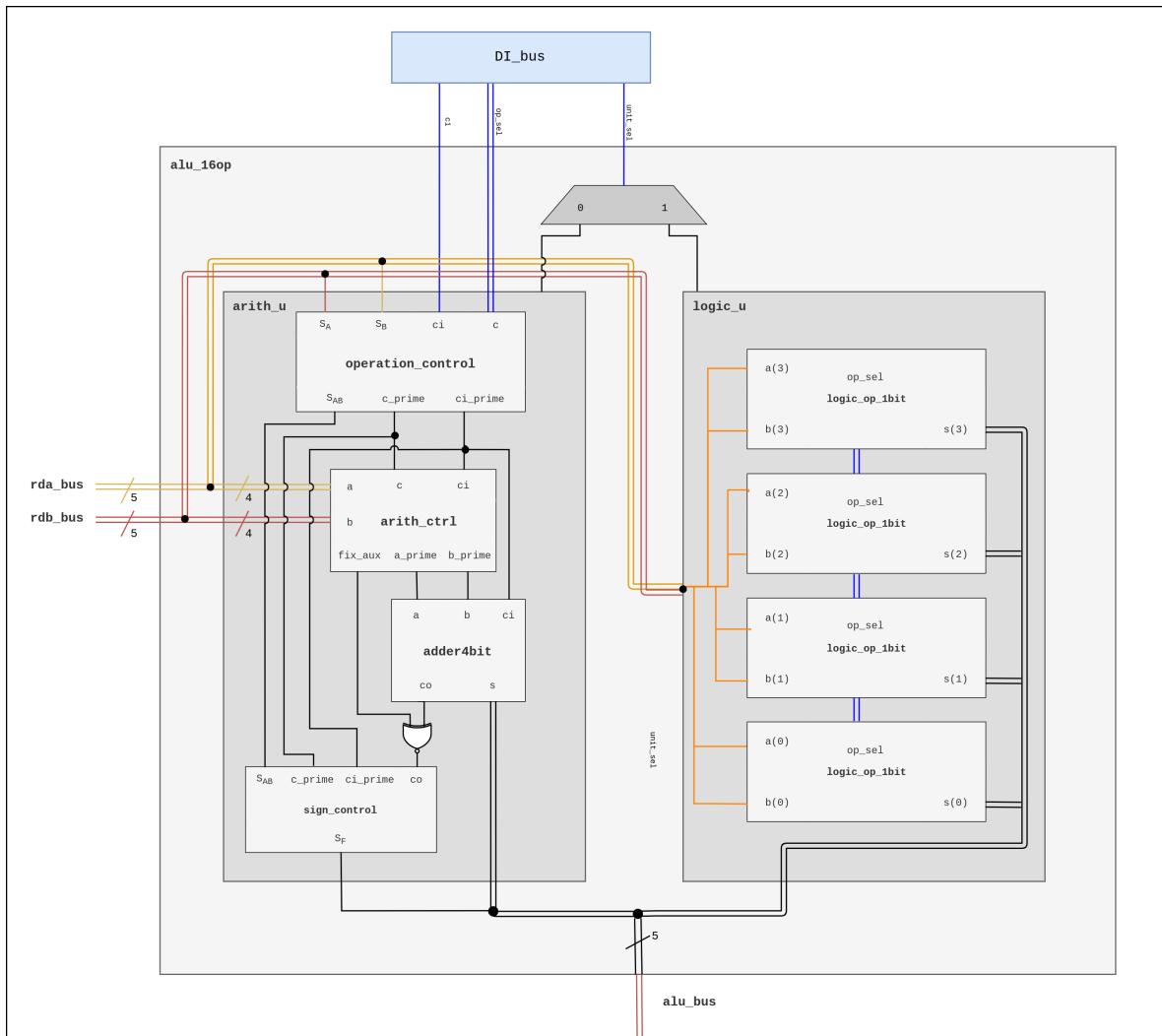


fig.3.0. Diagrama a bloques del Stage 2: ROM + PC + RI + DI.

## Manejo de signos

Para implementar esta función se agregaron dos bloques a la unidad aritmética, de control de operación `operation_control` y control de signo `sign_control`, estos, como se describe mas adelante, permiten manipular los operandos, el signo del resultado y la operación en sí, para lleva a cabo la instrucción solicitada con operandos de 4 bits con signo.



**fig.3.1.** Diagrama de la ALU de 16 Operaciones con signos

## Control de operación para suma/resta

Existe una equivalencia (Tabla 3.0) entre la suma con signos diferentes y la resta con signos iguales, es decir, según los signos de los operandos de entrada, la operación requerida se puede realizar considerando solamente la magnitud, la operación (suma o resta) equivalente correspondiente y finalmente realizando un ajuste en el signo del resultado de salida. La ventaja que esto supone se aprovecha para modelar el comportamiento del bloque de control de operación mostrado en la Tabla 3.1.

suma	$\iff$	resta
++	$\iff$	-+
+-	$\iff$	--
-+	$\iff$	++
--	$\iff$	+ -

**Tabla 3.0.** Equivalencia de operaciones de suma y resta entre valores de diferentes signos. La ventaja principal de esta equivalencia es la posibilidad de realizar una operación con operandos de signo diferente mediante una operación con operandos de signos iguales.

Recordando el formato de instrucción de 5 bits mencionado en la sección anterior, donde el valor de `op_sel & ci`, con `op_sel = c` para este caso, es igual a `000` correspondiente a la operación de suma y el valor `001` correspondiente a la operación de resta; tenemos que en función de los signos de los operandos de entrada A y B ( $S_A$  y  $S_B$  respectivamente), se modifica el bit  $ci$  para realizar la operación contraria; esto con la finalidad de ejecutar una operación equivalente como se muestra en la Tabla 3.0.

La siguiente tabla, muestra como salidas el bit de la operación prima  $ci\_prime$  y el signo equivalente  $S_{AB}$  de los operandos.

$c \& ci$	$S_A$	$S_B$	$ci\_prime$	$S_{AB}$
000	0	0	0	0
000	0	1	1	1
000	1	0	1	0
000	1	1	0	1
001	0	0	1	0
001	0	1	0	0
001	1	0	0	1
001	1	1	1	1

$ci\_prime$	$S_{AB}$
$ci \oplus S_A \oplus S_B$	$S_B$

**Tabla 3.1.** Tabla de verdad del control de operación para la suma y resta. Ecuaciones booleanas resultantes para las salidas  $ci\_prime$  y  $S_{AB}$ .

## Control de signo para suma/resta

Conociendo el signo de ambos ( $S_{AB}$ ), la operación equivalente u operación prima y el acarreo de salida  $co$  del sumador, definimos la tabla del bloque de control de signo donde la salida  $S_F$  indica si es necesario alterar el signo del resultado para mantener la coherencia de este. En la operación suma, por ejemplo, los signos pasan igual y en la resta depende del signo y del operando mayor.

<u><math>c\_prime</math></u>	$S_{AB}$	$co$	<u><math>S_F</math></u>
000	0	0	0
000	0	1	0
000	1	0	1
000	1	1	1
001	0	0	0
001	0	1	1
001	1	0	1
001	1	1	0

$S_F$
$[(not ci\_prime) and S_{AB}] or [ci\_prime and (S_{AB} \oplus co)]$

**Tabla 3.2.** Tabla de verdad del control de signo para la suma y resta y ecuación booleana de salida para  $S_F$ .

## Control de operación para transferencia, incremento y decremento

Para estas operaciones aritméticas restantes, se usaron las equivalencias entre el incremento de signo positivo y el decremento de signo negativo y viceversa, con el fin de definir el comportamiento del control de operación y signo en estas operaciones; cabe mencionar que la operación transferencia no requiere de esta función.

$S_{AB}$	$c \& ci$		<u><math>c\_prime</math></u>	<u><math>ci\_prime</math></u>	<u><math>S_F</math></u>
0	010	transferencia A	01	0	0
1	010	transferencia A	01	0	1
0	100	transferencia B	10	0	0
1	100	transferencia B	10	0	1
0	011	incremento A	01	1	0
0	101	incremento B	10	1	0
0	110	decremento A	11	0	0
0	111	decremento B	11	1	0
1	011	incremento A	11	0	1
1	101	incremento B	11	1	1
1	110	decremento A	01	1	1
1	111	decremento B	10	1	1

	transferencia/incremento A	transferencia/incremento B	decremento A y B
$c\_prime(0)$	$ci \text{ and } S_A$	1	$[(not ci) \text{ and } (not S_A)] \text{ or } ci$
$c\_prime(1)$	1	$ci \text{ and } S_B$	$(not ci) \text{ or } [ci \text{ and } (not S_B)]$
$ci\_prime$	$ci \text{ and } (not S_A)$	$ci$	$ci \text{ or } [(not ci) \text{ and } S_A]$
$S_{AB}$	$S_A$	$S_B$	$[(not ci) \text{ and } S_A] \text{ or } (ci \text{ and } S_B)$

**Tabla 3.3.** Tabla de verdad del control de signo para el incremento, decremento y transferencia. Ecuaciones de salida para  $c\_prime(0)$ ,  $c\_prime(1)$ ,  $ci\_prime$  y  $S_{AB}$ .

### Control de signo para la transferencia, incremento y decremento

$S_{AB}$	$c\_prime$	$S_F$
0	010	0
1	010	1
0	100	0
1	100	1
0	011	0
0	101	0
0	110	0
0	111	0
1	110	1
1	111	1
1	011	1
1	101	1

$S_F$
$S_{AB}$

**Tabla 3.4.** Tabla de verdad del control de signo para el incremento, decremento y transferencia y ecuación de salida para  $S_F$ .

## Simulación

Se realizaron 5 simulaciones para cada operación aritmética (suma, resta, transferencia, incremento y decremento) para visualizar el comportamiento de la ALU con la implementación descrita anteriormente. Las siguientes figuras muestran los resultados obtenidos.

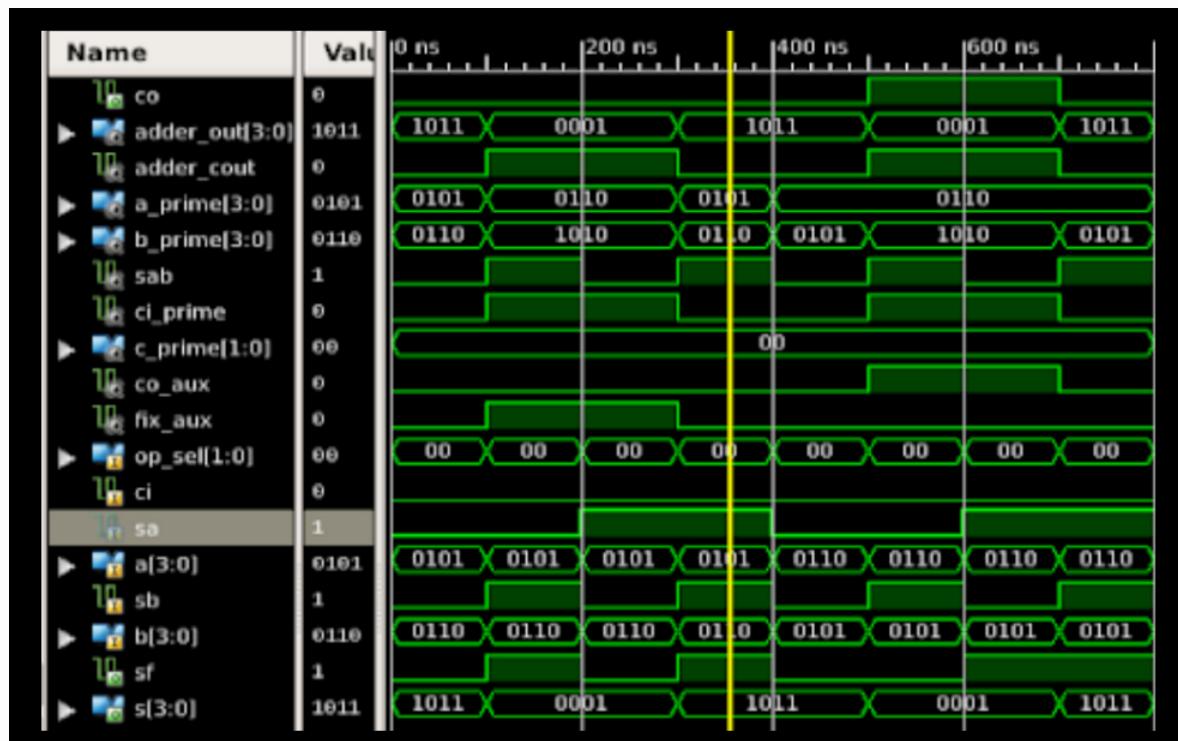


fig.3.2. Suma: los primeros 4 ciclos son con  $B > A$  y los siguientes 4 ciclos son para  $A > B$ , ambos con cada combinación de signos.

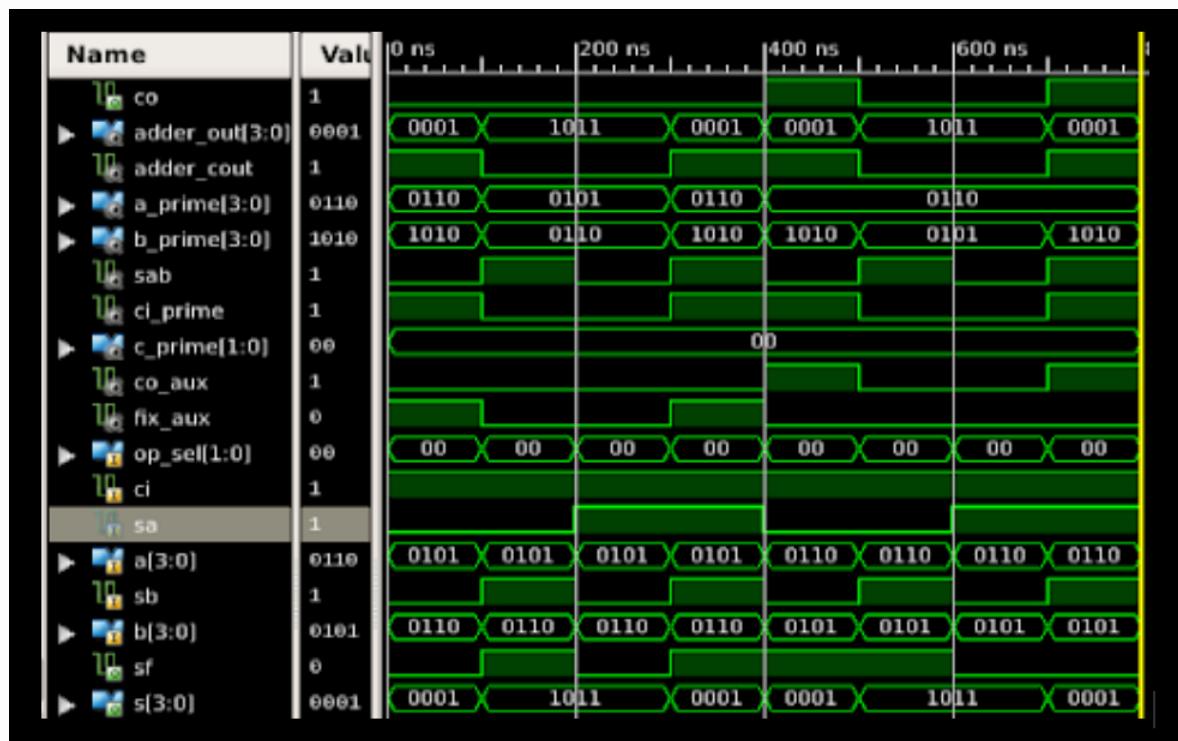


fig.3.3. Resta: los primeros 4 ciclos son con  $B > A$  y los siguientes 4 ciclos son para  $A > B$ , ambos con cada combinación de signos.

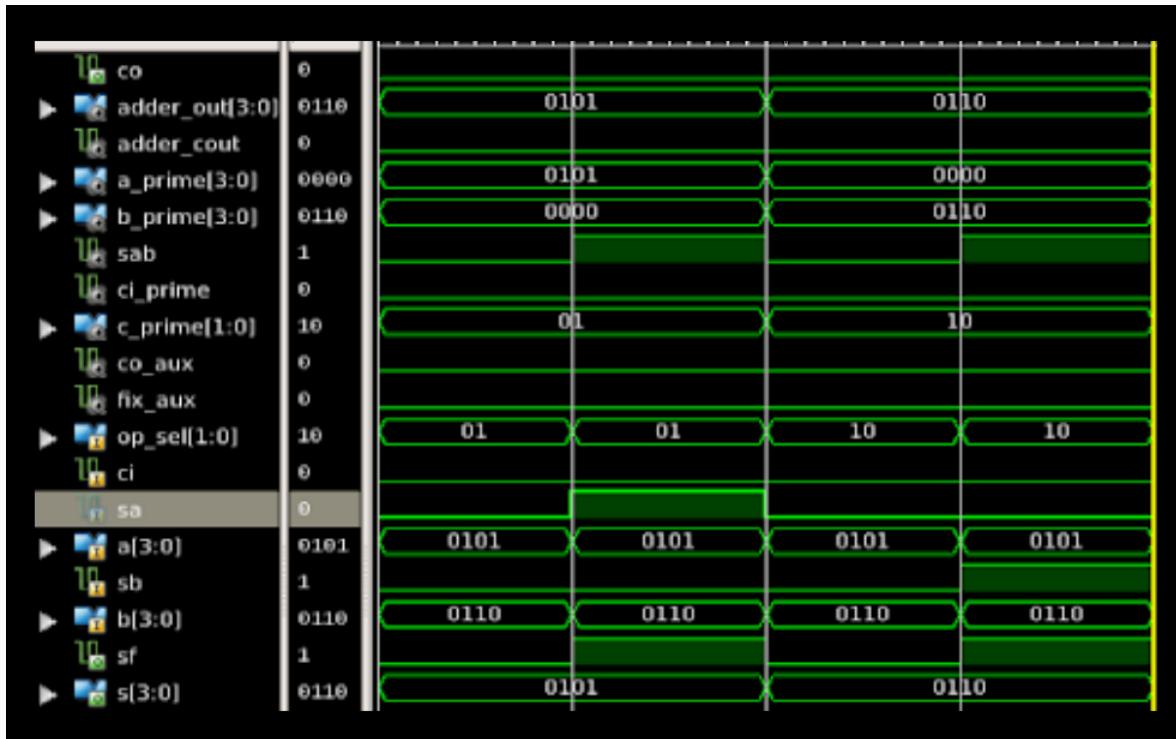


fig.3.4. Transferencia: los primeros 2 ciclos son la operación transferencia de A con signo positivo y negativo, los siguientes 2 son para la transferencia de B.

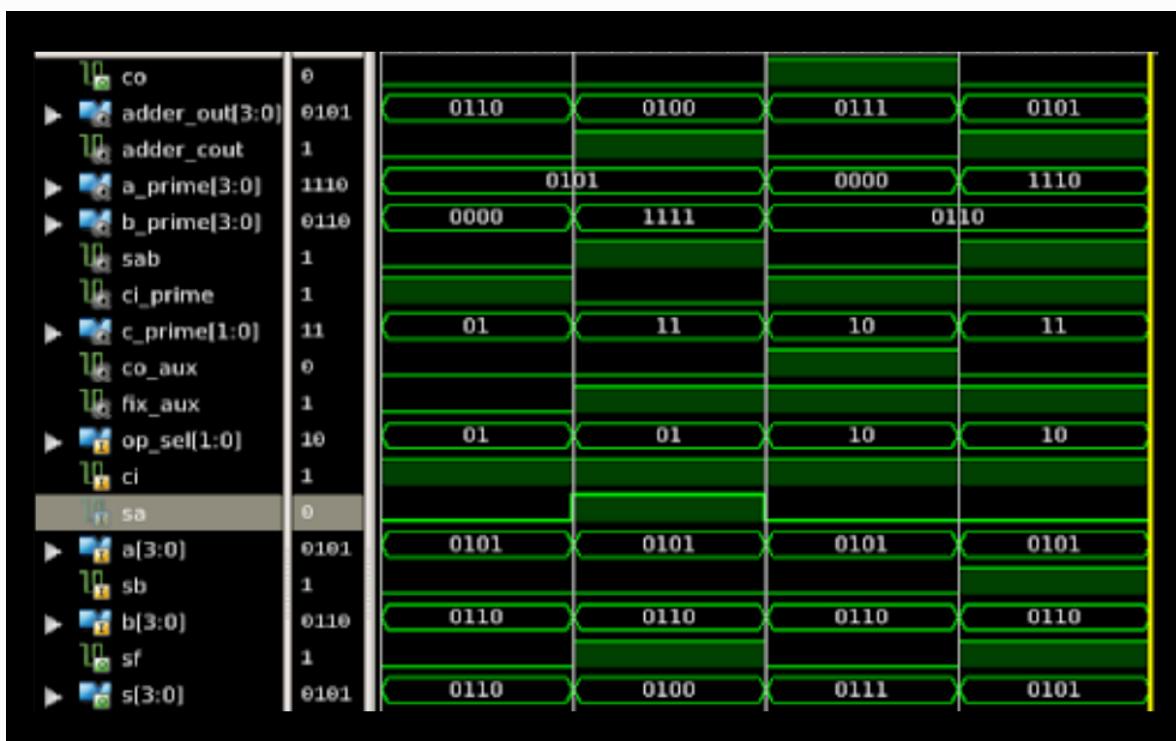


fig.3.5. Incremento: los primeros 2 ciclos son la operación incremento de A con signo positivo y negativo, los siguientes 2 son para la incremento de B.

co	1				
► adder_out[3:0]	0111	0100	0110	0101	0111
► adder_cout	0				
► a_prime[3:0]	0000	0101	1110	0000	
► b_prime[3:0]	0110	1111	0000	0110	
► sab	1				
► ci_prime	1				
► c_prime[1:0]	10	11	01	11	10
► co_aux	1				
► fix_aux	1				
► op_sel[1:0]	11	11	11	11	11
► ci	1				
► sa	0				
► a[3:0]	0101	0101	0101	0101	0101
► sb	1				
► b[3:0]	0110	0110	0110	0110	0110
► sf	1				
► s[3:0]	0111	0100	0110	0101	0111

fig.3.6. Decremento: los primeros 2 ciclos son la operación decremento de A con signo positivo y negativo, los siguientes 2 son para la decremento de B

## ROM

Una memoria ROM es un dispositivo digital que permite almacenar y acceder a la información aunque se apague el dispositivo, en un microcontrolador la memoria ROM se usa para almacenar las instrucciones y a veces variables que no cambian. Para que el microcontrolador pueda funcionar independientemente es necesario incorporarle un modulo de memoria que nos permitirá ejecutar las instrucciones secuencialmente al acceder a una dirección mediante el contador de programa.

Una memoria ROM consiste en  $2^n$  registros de m bits, m multiplexores de n bits de selección que permite acceder a los m bits de un registro, a la linea de selección de la memoria ROM se le conoce como dirección de memoria, ya que cada registro corresponde con un valor de esta dirección de memoria, en la figura 3.7 se muestra el esquema de la memoria ROM utilizada.

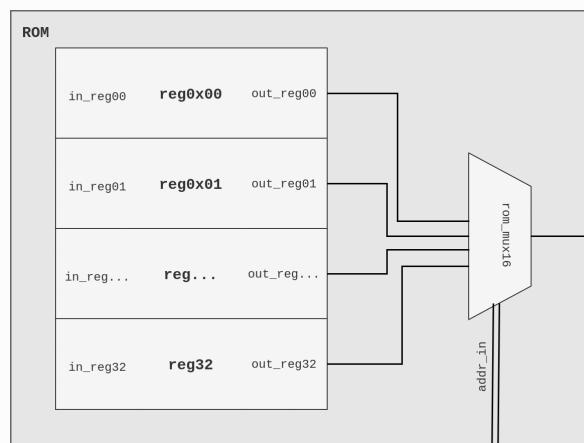


fig.3.7. Diagrama a bloques del módulo de memoria ROM.

En este caso se uso una memoria ROM con 32 registros de 16 bits y una linea de selección de 5 bits `addr_in`, se selecciono el tamaño de 32 registro ya que esta memoria es unicamente para pruebas, los registros son de 16 bits que corresponde con el formato de instrucción del microcontrolador.

## Registro de instrucción RI + Decodificador de instrucción

El registro de instrucción almacena y entrega la instrucción que se obtiene de la memoria ROM al activarse el ciclo de reloj correspondiente al decodificador de instrucción, este se encarga de obtener los datos de la instrucción y preparar los registros necesarios para realizar la operación en el ciclo de la ALU, que puede ser activar la entrada del contador de programa e ingresar una nueva dirección o colocar las entradas en los registros de datos así como la operación a realizar en la ALU. Este modulo se puede representar como un demultiplexor que recibe la instrucción y la envía al contador de programa o a la ALU.

## Contador de programa PC

El contador de programa (PC) es un registro del procesador de un computador que indica la posición donde está el procesador en su secuencia de instrucciones. Para que el microcontrolador pueda funcionar independientemente es necesario este componente, puesto que nos permite acceder a una dirección de memoria y ejecutar las instrucciones.

Un contador de programa simplemente realiza la función de contador, llevando el registro de las instrucciones que se están ejecutando y cuando se manda a llamar el `in_enable`, realiza una instrucción especifica, en la figura 3.8 se muestra el diagrama del componente implementado.

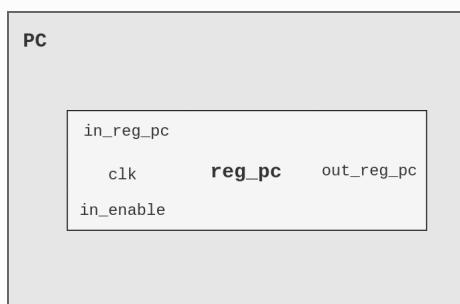


fig.3.8. El puerto entrada `in_reg_enabe` recibe un señal en alto cuando se debe hacer un "salto" a la dirección indicada en puerto de entrada de 5 bits `in_reg_pc`. El puerto de salida `out_reg_pc` se conecta mediante un bus a la ROM para obtener de la dirección especificada la instrucción de 16 bits. El puerto de entrada `clk` recibe la señal del E del GCM para secuenciar de forma correcta el funcionamiento del PC.

## Instrucción de 16 bits

El funcionamiento en conjunto de las dos etapas del microprocesador, implica el manejo de instrucciones con longitud de 16 bits. Dicha instrucción se conforma de 7 campos de longitud fija, que ademas de contener la instrucción de 5 bits discutida anteriormente, integra el valor de los operandos A y B y en el último bit se define un control para el flujo de los los datos; este control (bit 15) en bajo indica que los campos de datos A y B se consideran operandos de 5 bits cada uno y son enviados a la ALU para su operación, en el caso contrario de tener un valor alto en este bit, ambos campos se

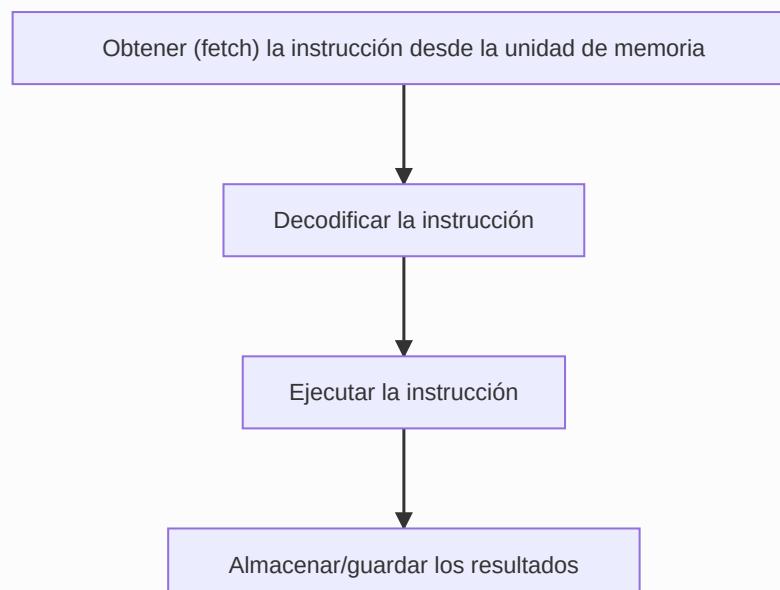
consideran una dirección de 5 bits, la cual se envia al contador de programa para que ejecute un "salto" a dicha dirección dentro de los registros de la memoria ROM.

15	14	13	11-12	10	9-5	4-0
alu	regmux_sel	unit_sel	op_sel	carry_in	dato B	dato A

Tabla 3.5. Formato de instrucción de 16 bits

## GCM

Como se ha mencionado anteriormente y como se ha observado en las simulaciones del **micro\_stage1**, el GCM es el componente encargado de controlar el proceso secuencial en la ejecución de una instrucción en un programa. Podemos definir de forma elemental y resumida como el componente en el cual a partir de una señal de entrada de un oscilador, envía a la salida un conjunto de señales secuenciadas a los componentes del microprocesador. Estas señales de salida se pueden identificar dentro de 4 acciones principales:



En el diseño del microprocesador establecimos que la segmentación de este se da en función del conjunto de señales provenientes del GCM A-E. La siguiente tabla muestra el identificador de la señal y la acción que se realiza durante su tiempo en alto.

Señal	Acción
A	Cargar instrucción desde la memoria en el registro de instrucción RI
B	Cargar los datos de entrada a la ALU en el registro RD
C	Almacenar el resultado de la operación procedente de la ALU en el registro ACC
D	Cargar en el registro ACM el valor almacenado en el registro ACC
E	Aumentar o modificar el contador del programa

# Descripción

La descripción del **Generador de Ciclo de Máquina** se puede hacer mediante un contador de anillo de 5 bits cuya comportamiento se describe en la Tabla 3.5. A la salida de este contador se conecta un codificador de 5 entradas a 5 salidas el cual se encarga de enviar el bit encendido del estado actual del contador al puerto de salida correspondiente a la salida A-E, según sea el caso.

oscilator	gcm_count	gcm_count'	Salida
1	00001	00010	A
1	00010	00100	B
1	00100	01000	C
1	01000	10000	D
1	10000	00001	E

Tabla 3.6. Tabla de estados del contador de anillo utilizado en el GCM.

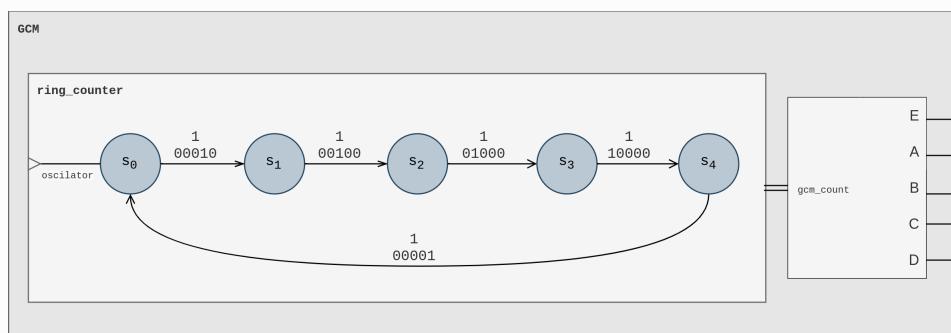


fig.3.9. Diagrama a bloques del GCM.

## Simulación

La siguiente figura muestra la simulación correspondiente al GCM. Se muestran los ciclos a diferentes escalas de tiempo con un periodo de  $400\text{ps}$ .

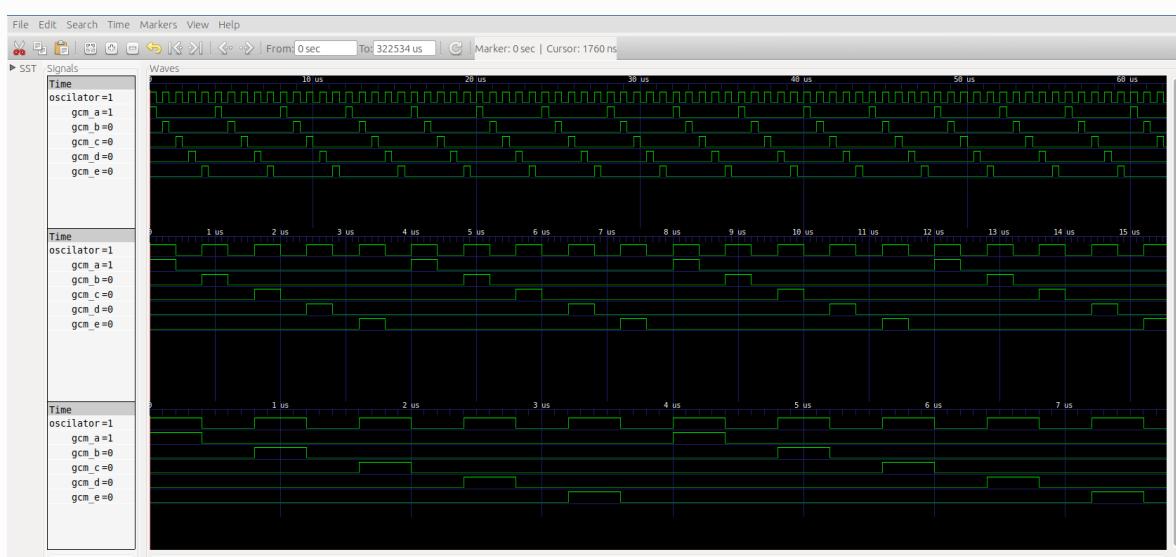


fig.3.10. Simulación del funcionamiento del GCM con las 5 señales A-E.



## IV: LagartijaX4

# IV: LagartijaX4

Para observar el funcionamiento del microprocesador se monta una simulación del componente [lagartijax4](#); dicho componente está integrado por las dos etapas Stage 1 y Stage 2 conectadas dentro del [micro\\_core](#) el cual a su vez recibe las señales provenientes del [gcm](#). El entorno de simulación muestra las señales, entradas y salidas de los componentes notables de cada uno de los módulos con el fin de mostrar el funcionamiento del microprocesador desde diferentes niveles de diseño.

## Instrucciones de prueba

Se define en los registros de la memoria ROM una secuencia de instrucciones para realizar las 16 operaciones tanto aritméticas como lógicas disponibles sobre diferentes operandos, para posteriormente realizar un salto entre instrucciones como se detallará en las siguientes tablas.

registro	instrucción	registro	instrucción
0x0	0000000100000011	0x10	0011100110001111
0x1	0000010110000100	0x11	0111110111101100
0x2	0100000000011011	0x12	0111000010101111
0x3	0000100011101101	0x13	1000000000011001
0x4	0001000011101101	0x14	1000000000011010
0x5	0000110011000001	0x15	1000000000011011
0x6	0101010011000001	0x16	1000000000011100
0x7	0001101111000101	0x17	1000000000011101
0x8	0001111111000101	0x18	10000000000011110
0x9	0100010000010101	0x19	10000000000010100
0xA	0010000011001010	0x1A	10000000000010101
0xB	0010010011101111	0x1B	10000000000010110
0xC	0110100101000110	0x1C	10000000000010111
0xD	0110110000000101	0x1D	10000000000011000
0xE	0011000110100100	0x1E	0111000010101111
0xF	0011010010001101	0x1F	10000000000010010

a)

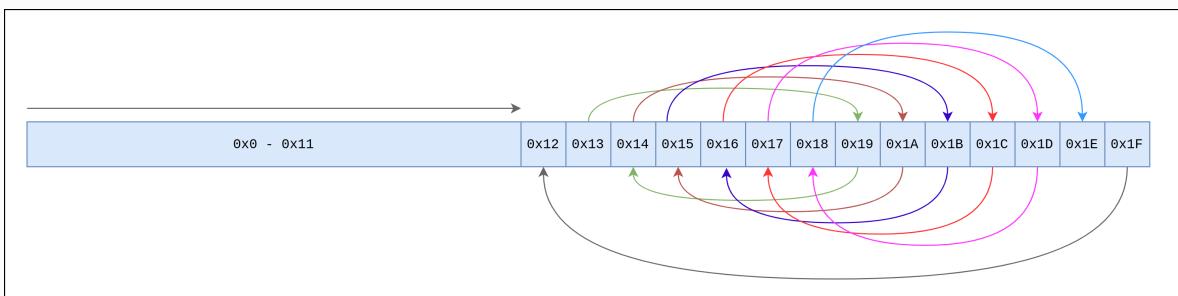
registro	alu/pc	unidad	operando	operación	B	A	resultado	registro	alu/pc	unidad	operando	operación	B	A	resultado
0x0	ALU	aritmética	B	suma	8	3	01011	0x10	ALU	aritmética	B	decremento A	0	14	01101
0x1	ALU	aritmética	B	resta	12	4	01000	0x11	ALU	aritmética	B	transferencia B	5	0	00101
0x2	ALU	aritmética	ACM	suma	0	-11	10011	0x12	ALU	aritmética	B	transferencia B	-5	0	10101
0x3	ALU	aritmética	ACM	transferencia A	0	13	01101	0x13	ALU	lógica	B	transferencia B	10	5	01111
0x4	ALU	aritmética	ACM	transferencia B	0	13	01101	0x14	ALU	lógica	B	incremento B	3	6	01010
0x5	ALU	aritmética	B	incremento A	6	1	00010	0x15	PC	-	-	-	-	1E	0xE
0x6	ALU	aritmética	B	incremento B	6	1	00111	0x16	ALU	aritmética	B	suma	0	9	-
0x7	ALU	aritmética	B	decremento A	7	5	00100	0x17	ALU	aritmética	B	suma	0	8	-
0x8	ALU	aritmética	B	decremento B	7	5	00110	0x18	ALU	aritmética	B	suma	0	7	-
0x9	ALU	lógica	B	decremento B	0	5	01010	0x19	ALU	aritmética	B	suma	0	6	-
0xA	ALU	aritmética	B	suma	7	8	01111	0x1A	ALU	aritmética	B	suma	0	5	-
0xB	ALU	aritmética	B	suma	10	-2	01000	0x1B	ALU	aritmética	B	suma	0	4	-
0xC	ALU	aritmética	B	resta	-10	2	11100	0x1C	ALU	aritmética	B	suma	0	3	-
0xD	ALU	aritmética	B	incremento A	0	-14	11101	0x1D	ALU	aritmética	B	suma	0	2	-
0xE	ALU	aritmética	B	incremento A	0	15	00000	0x1E	PC	-	-	-	-	1	0x1
0xF	ALU	aritmética	B	decremento A	0	-14	11111	0x1F	ALU	aritmética	B	suma	0	0	-

b)

**Tabla 4.0.** Ejemplo de un conjunto de 32 instrucciones. a) Instrucciones de 16 bits en binario. b) Instrucciones decodificadas.

Para el caso de la tabla anterior Tabla 4.0.b, la primer columna corresponde a la dirección de la instrucción dentro de la ROM. La siguiente columna indica si los datos son enviados a la ALU o al PC. De ser enviados a la ALU la columna *unidad* indica si se trata de una operación lógica o aritmética, la columna *operando* muestra el registro que se considera como segundo operando, ya sea el valor B leído en los bits 5-9 o el almacenado en el registro acumulador ACM. La columna *operación* muestra específicamente la operación a realizar. La columna B y A muestra el valor decimal de los operandos leídos en la instrucción, si los datos son enviados al PC, no se considera el operando B y en la columna A se muestra la dirección de memoria a la que se debe mover el PC.

Como se ha observado en las tablas anteriores, el conjunto de instrucciones realiza cada una de las operaciones disponibles en la ALU, tanto aritméticas (0x0 - 0x9) como lógicas (0xA - 0x18) y alternando valores positivos y negativos y a su vez especificando como segundo operando el valor almacenado en un momento dado en el acumulador (0x2, 0x6, 0x9, 0xC, 0xD, 0x11 y 0x12); posteriormente se realiza una serie de "saltos" entre registros de la ROM para finalmente regresar a la instrucción 0x12 y repetir el ciclo (fig.4.0.).



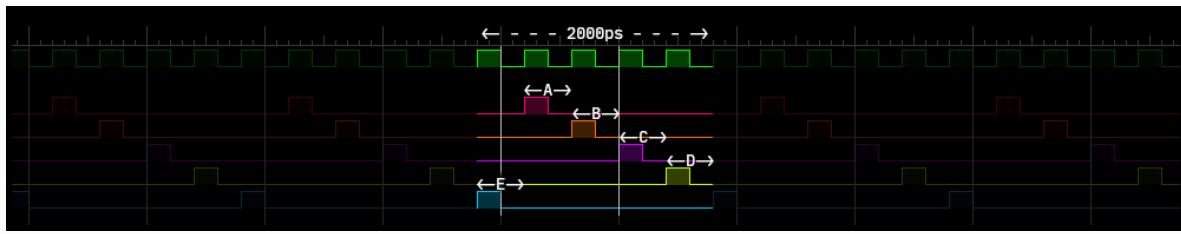
**fig.4.0.** Trayectoria del contador de programa en la memoria ROM.

## Simulación

Manteniendo la convención de la frecuencia de  $2.5GHz$ , y por ende una señal de reloj con un periodo de  $400ps$ , tenemos que por cada instrucción, al microprocesador le llevará

$$t_{instrcn} = t_A + t_B + t_C + t_D + t_E = 5 \cdot T_{clk} = 5 \cdot 400ps = 2000ps$$

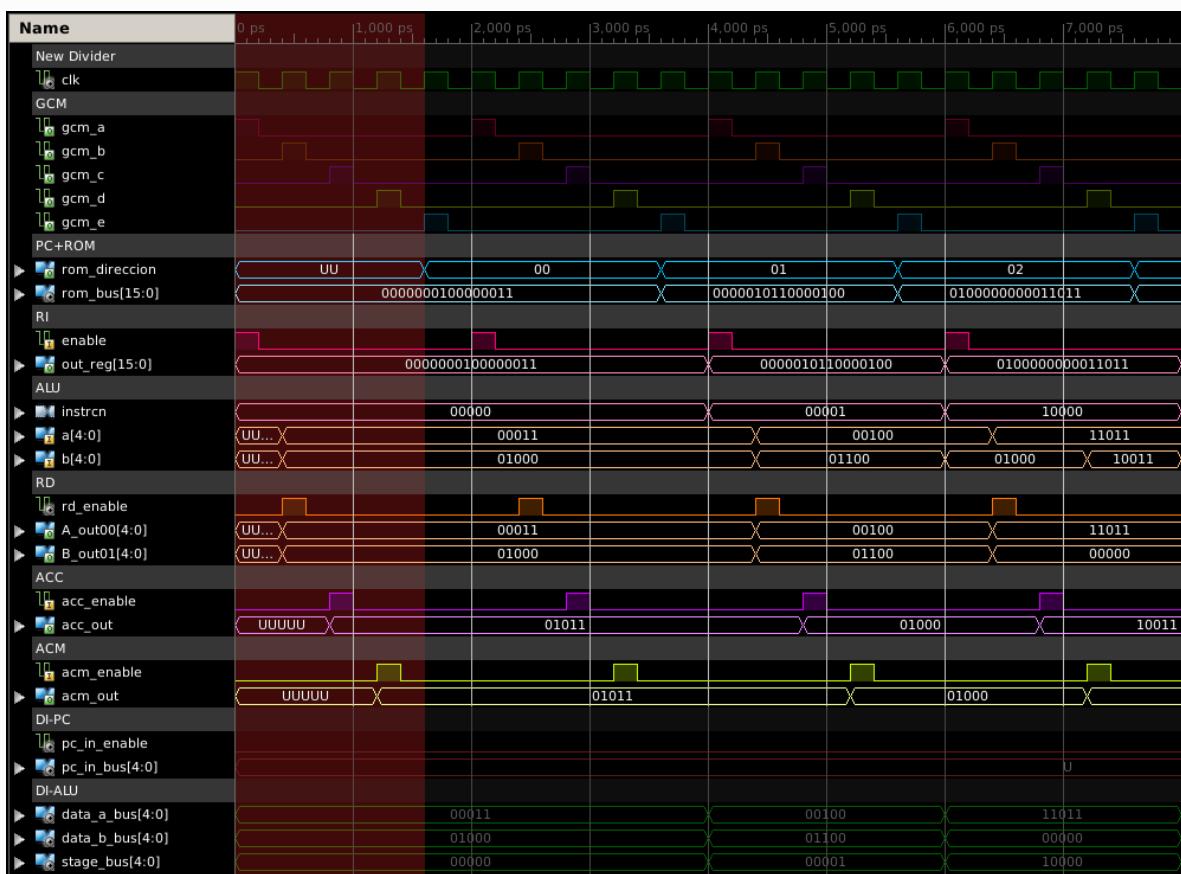
como lo ilustra la siguiente figura:



**fig.4.1.** Ejemplo de tiempo de ejecución de una instrucción y los respectivos pulsos del GMC. Observamos que la señal E se considera como la primera dentro del ciclo, esto debido a que es la señal que permite obtener una instrucción desde la ROM.

**Nota:** Debido a que independientemente de que el ciclo de la instrucción comience con la señal E, la primer señal enviada por el GCM es la señal A, lo cual tendrá como consecuencia un tiempo "muerto" de 1600ps al realizar la ejecución de la primera instrucción.

La siguiente serie de figuras muestra la simulación del programa descrito anteriormente, para las operaciones aritméticas y lógicas se destaca el valor de los operandos y el resultado cargado en los registros ACC y ACM. Para observar el salto entre instrucciones se destaca el valor del bus de dirección del PC.



**fig.4.2.** Instrucciones: **0x0** - **0x2**. La parte sombreada en rojo corresponde a los 1600ps mencionados en la fig.4.1.

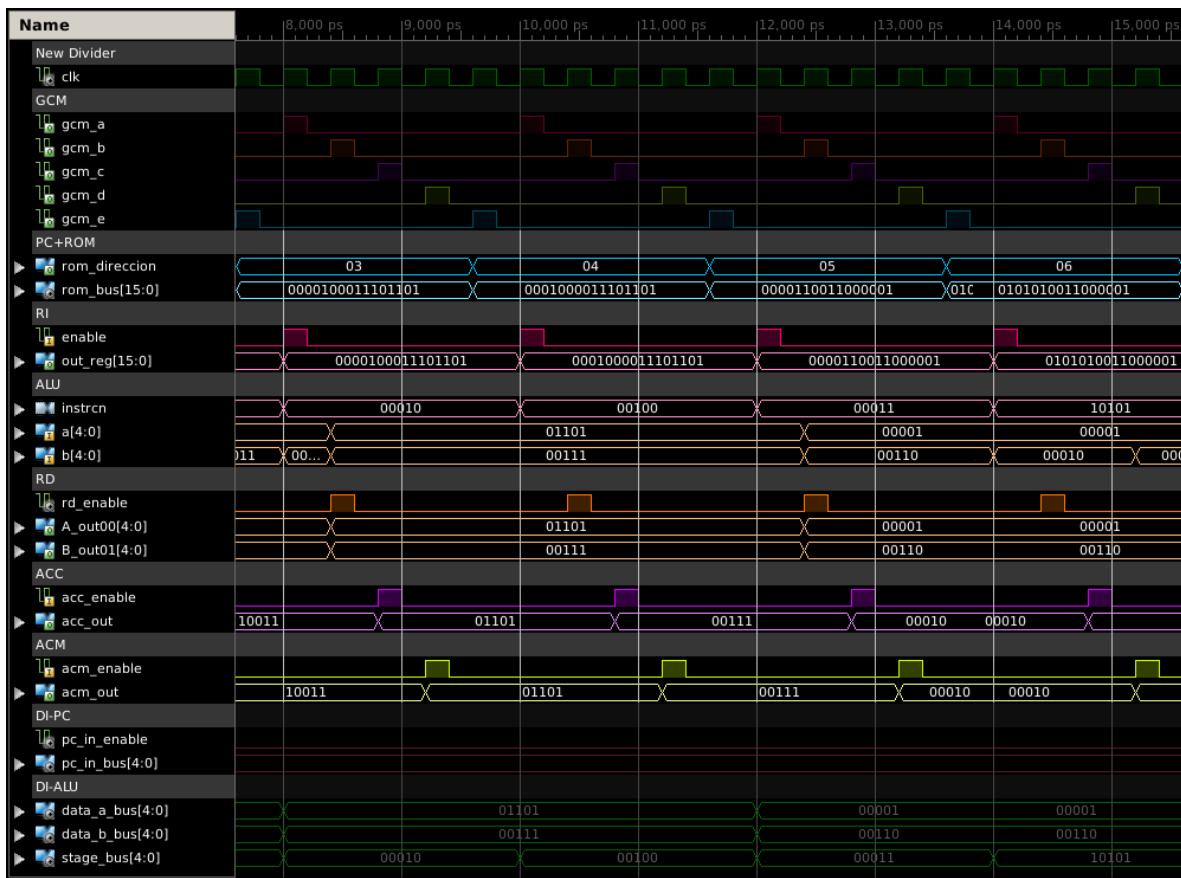


fig.4.3. Instrucciones: 0x3 - 0x5 .

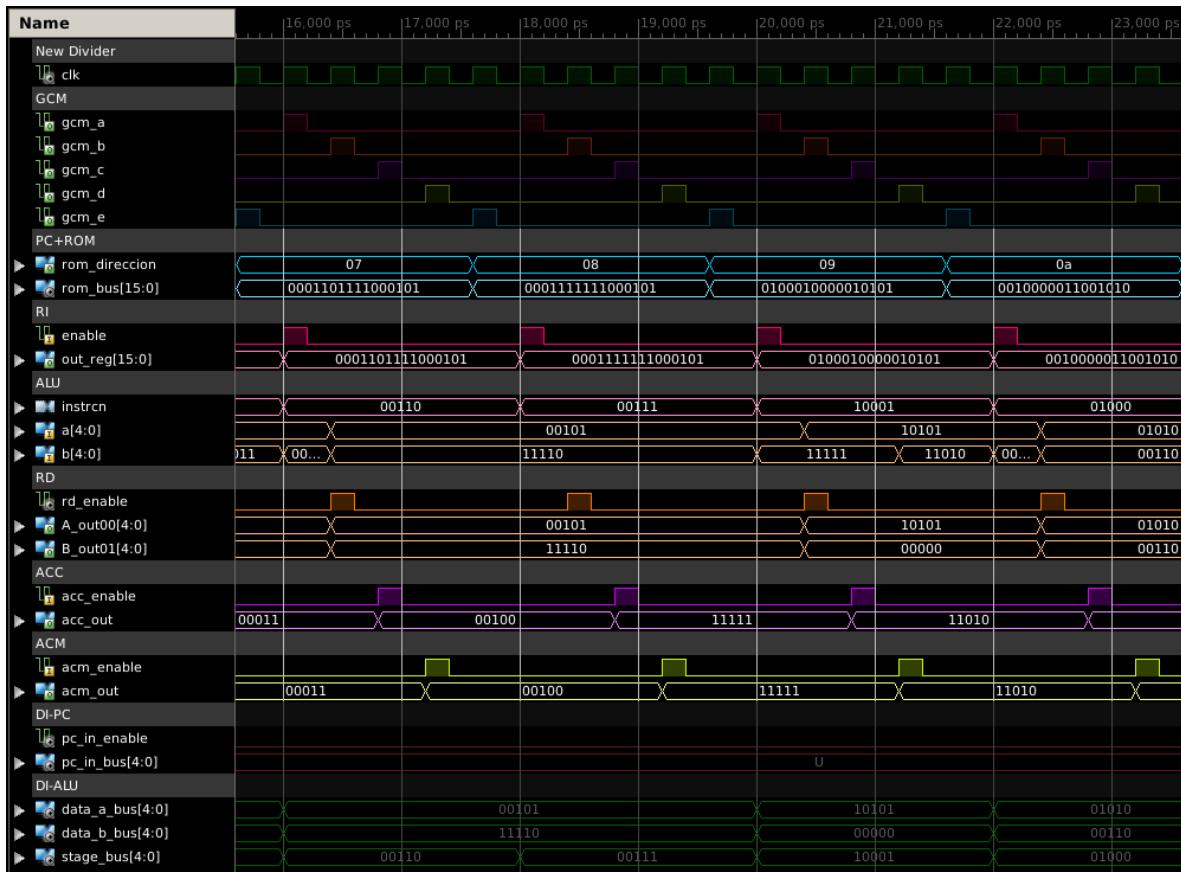


fig.4.4. Instrucciones: 0x7 - 0xA .

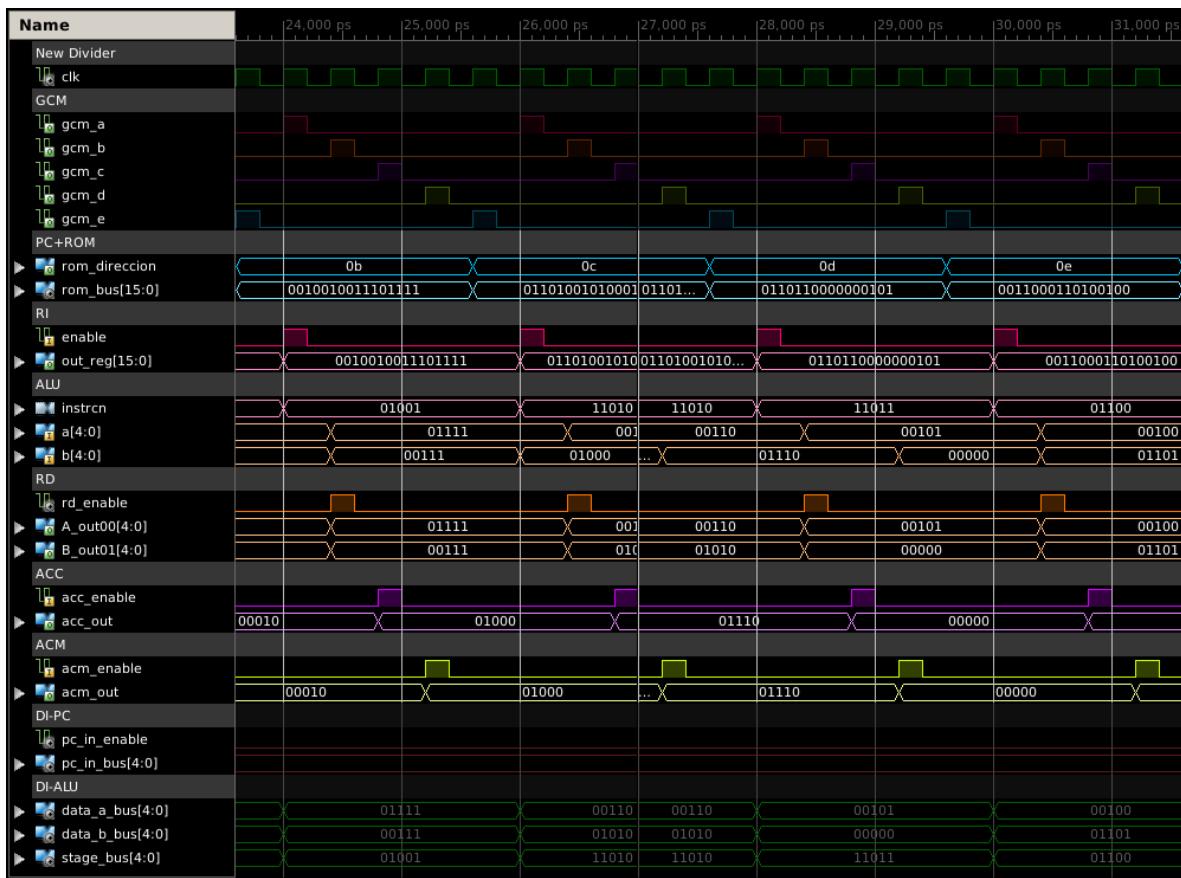


fig.4.5. Instrucciones: **0xB** - **0xE**.

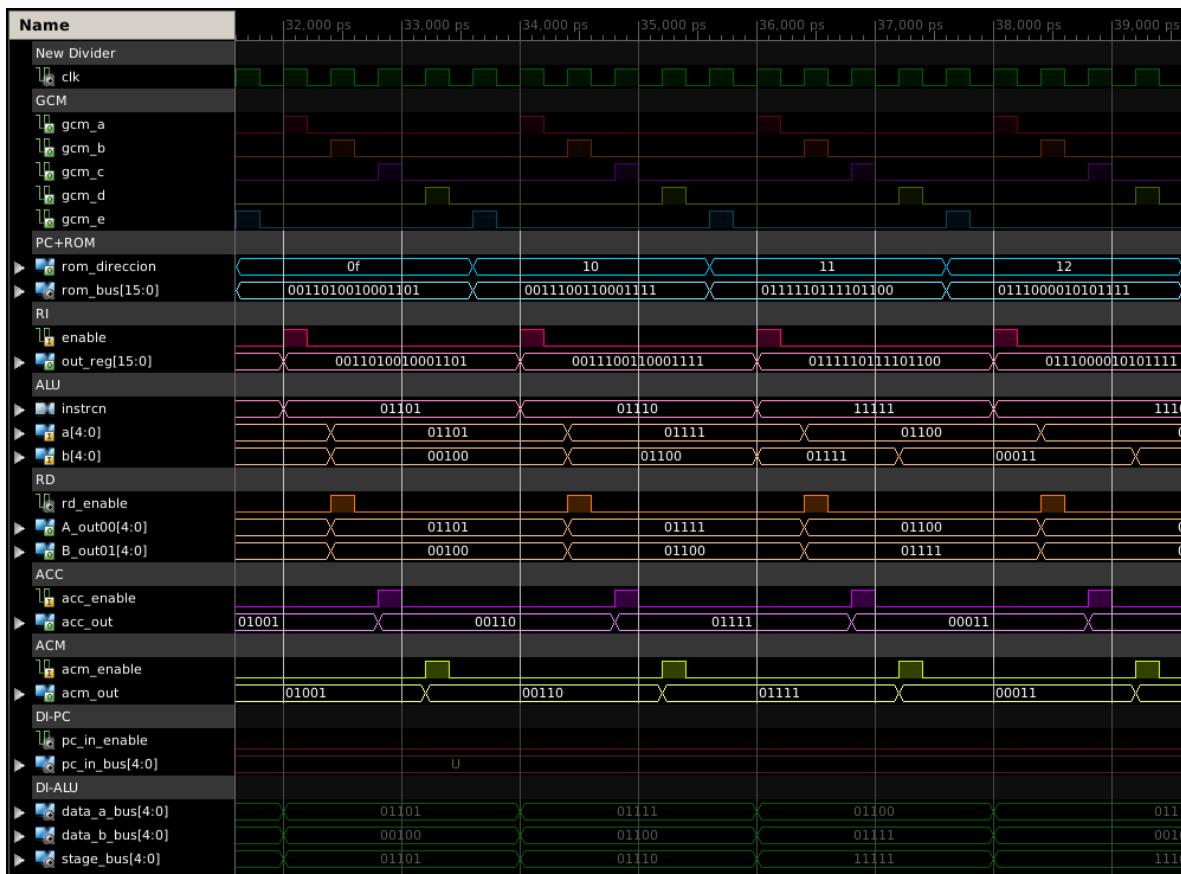
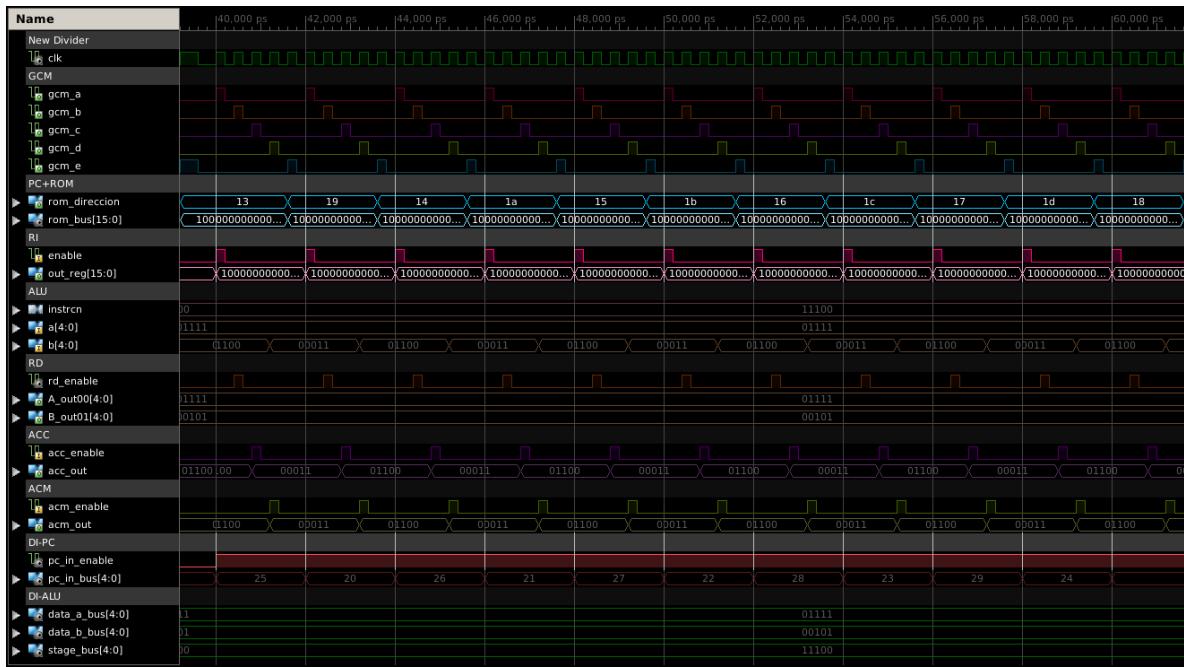
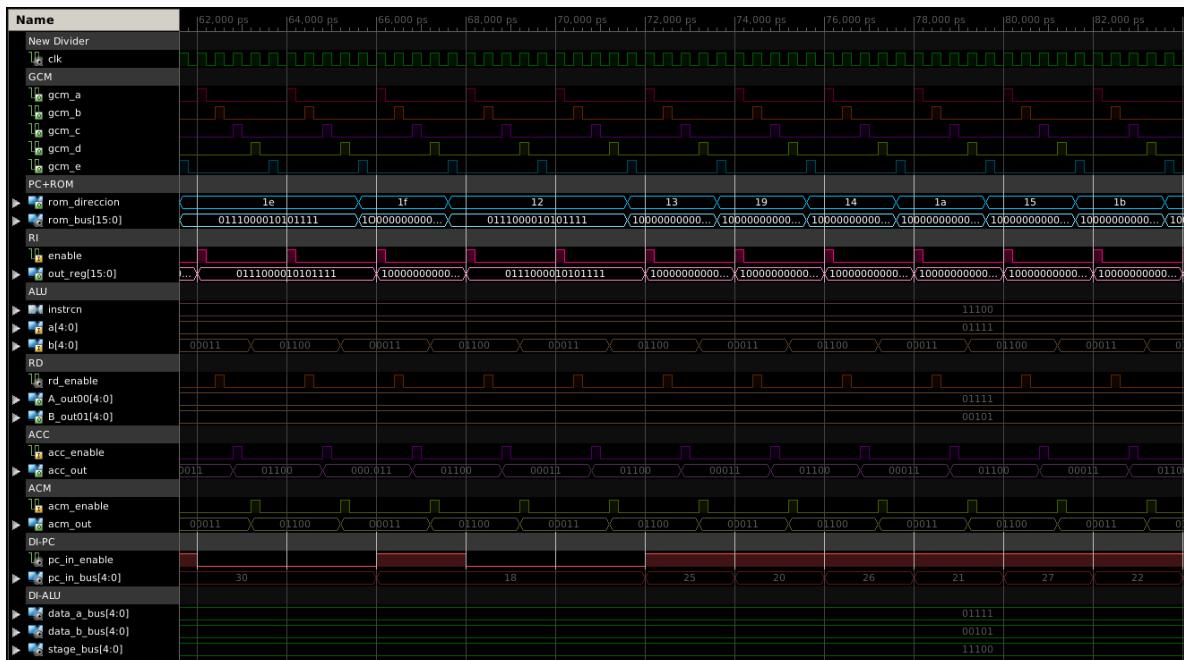


fig.4.6. Instrucciones: **0xF** - **0x12**.



**fig.4.7. Instrucciones: 0x13 - 0x18.** Se observa el "salto" entre instrucciones ilustrado en la fig. 4.0. La señal `pc_in_enable` se activa para indicar al PC un salto a la dirección indicada en el `pc_in_bus`.



**fig.4.8. Instrucciones: 0x1e - 0xb.** Una vez que se alcanza la dirección `0x1e` se realiza la operación, el contador de programa pasa a la siguiente instrucción `0x1f` para después volver a `0x12` y repetir el ciclo iniciado en `0x13`.

## Conclusión

El objetivo del diseño e implementación del Microprocesador de 4 bits "LagartijaX4", se centra principalmente en analizar los componentes mínimos que integran un procesador así como las fases que implica la ejecución de una simple instrucción; esto se pretende lograr con los diagramas, tablas de verdad, descripciones, simulaciones y código que integran el presente proyecto de diseño. La segmentación del proyecto en fases se ha hecho con la finalidad de poder agregar funcionalidades, módulos o componentes de forma efectiva; tal es el caso de la `micro_stage3` en la que el equipo

nos encontramos trabajando actualmente, la cual tiene como objetivo incluir un componente de memoria RAM en el microprocesador y de esta forma hacer la transición de la arquitectura Von Neuemann a arquitectura Harvard.

Finalmente, hablando como equipo, podemos decir que el proceso que ha supuesto llevar a cabo este diseño e implementación ha requerido de la compilación y estudio de una porción considerable de teoría, metodologías, técnicas, estructuras, dispositivos y demás conceptos relacionados con la arquitectura de computadoras y por ende el Diseño Digital, lo cual tiene como consecuencia una mayor compresión de lo que es la computación hoy en día. El familiarizarse con el trabajo de un microprocesador a diferentes escalas trae consigo una ventaja en todos los aspectos que conforman nuestra formación, pues con ello los criterios y consideraciones que se tomen a la hora de desenvolvernos profesionalmente en cualquier área de la carrera, estará influenciado por este conocimiento.

---

**Escuela Superior de Cómputo - Arquitectura de computadoras**

# LAGARTIJAX4

INSTITUTO POLITÉCNICO NACIONAL



# Bibliografía y Referencias electrónicas

- [1] **Parhami Behrooz.** Arquitectura de Computadoras, De los microprocesadores a las supercomputadoras, 1ra. Edición. McGraw Hill, México, 2007.
- [2] **Fernando Pardo Carpió, José A. Boluda Grau.** VHDL Lenguaje para síntesis y modelado de circuitos. RA-MA Editorial, España, 1999.
- [3] **M. Morris Mano.** Diseño digital. 3ra Edición. PEARSON EDUCACIÓN, México, 2003.
- [4] **Luc Boulesteix.** Respuesta a: What is the fastest von Neumann architecture or Harvard architecture? en Quora.com.

<https://www.quora.com/What-is-the-fastest-von-Neumann-architecture-or-Harvard-architecture>

- [5] **Pablo Espeso.** CISC frente a RISC, una batalla en blanco y negro en Xataka.com.

<https://www.xataka.com/componentes/cisc-frente-a-risc-una-batalla-en-blanco-y-negro>

# Anexo A: Nomenclatura de dispositivos

## Circuitos básicos

### Multiplexor

```
1 mux[elementos][tamaño][selector]
2
3 elementos: N número de señales de entrada
4 tamaño:     Tamaño en bits de las entradas y salida
5 selector:   Tamaño en bits de selector = ⌈log2(elementos)⌉
```

### Demultiplexor

```
1 dmux[elementos][tamaño][selector]
2
3 elementos: N número de señales de salida
4 tamaño:     Tamaño en bits de las entradas y salida
5 selector:   Tamaño en bits de selector = ⌈log2(elementos)⌉
```

### Contador

```
1 cont[tipo][tamaño][inicio][fin]
2
3 tipo:
4     a      ascendente
5     d      descendente
6 tamaño:    N bits
7 inicio:    Posición inicial (inicio<2^N)
8 fin:       Posición final (inicio<fin<2^N)
```

### Decodificador

```
1 deco[tamaño entrada|código de entrada][tamaño salida|código de salida]
2
3 tamaño: Tamaño en bits
4 código: codificación (gray, 7seg, etc)
```

# Dispositivos de memoria

## Flip Flops

```
1 ff[tipo]
2
3 tipo:
4   D  0
5   T  1
6   SR 2
7   JK 3
```

## Registros

```
1 reg[tipo][tamaño]
2
3 tipo:
4   PP  paralelo-paralelo  0
5   PS  paralelo-serie    1
6   SP  serie-paralelo    2
7   SS  serie-serie      3
8
9 tamaño: N bits
```

## Banco de registros

```
1 bank[elementos][tamaño][sálida]
2
3 elementos: N registros, N>1
4 tamaño: 2^M bits
5 sálida: K registros, N>=1
```