Facultad de Ingeniería - UNLP
E0301 – Introducción a los Sistemas Lógicos y Digitales
Curso 2023 – Trabajo integrador

Fecha de entrega- La presente tarea tiene fecha máxima de entrega de acuerdo a lo publicado en la plataforma Moodle.

Lineamientos generales- Se sugiere realizar la entrega en un único archivo (preferiblemente en formato PDF). Si se incluyen fotos de lo resuelto en papel, procurar que las mismas estén en foco y con buen contraste de modo de simplificar el proceso de corrección.

Evaluación- La tarea será evaluada y los alumnos deberán defender el trabajo realizado en la fecha designada por la cátedra (la cual se informará con anticipación).

Unidad Aritmético Lógica

Como se vió en la teoría, una Unidad Aritmético Lógica o ALU, permite realizar distintas operaciones aritméticas (Suma / Resta / Negativo de un número) y Lógicas tanto entre dos operandos como en uno solo (AND, OR, XOR, NOT, LSHL, RSHL, etc.).

Además del resultado solicitado según la operación solicitada, la ALU provee de otras salidas que indican si se produjo un acarreo, o un overflow, además de indicar si el resultado es cero o negativo. Estas salidas se describen habitualmente como “Flags”.

Dependiendo de la arquitectura del sistema, se asocian *registros* para almacenar los operandos, resultados y Flags. En la Figura 1, se muestra que el registro A almacena uno de los operandos de entrada y que luego de realizada la operación almacena el resultado obtenido.

Este registro normalmente se denomina acumulador (ACC). El registro B debería contener el segundo operando si es necesario para la operación y un tercero almacenar los flags.

Las operaciones que se realizan sobre los operandos, se seleccionarán mediante comandos que consistirán en patrones de bits asociados acada operación (selección de operación) .

Debe tenerse en cuenta que los registros son Sincrónicos, es decir que hay una señal de reloj que permite su carga en instantes determinados, y además tienen una

o más señales de control (Read, Write, Clear, etc) que permiten su lectura y escritura.

En la Figura 1, hay dos señales externas: **LoadA** y **LoadB** que permiten la carga de los registros desde la memoria**.**

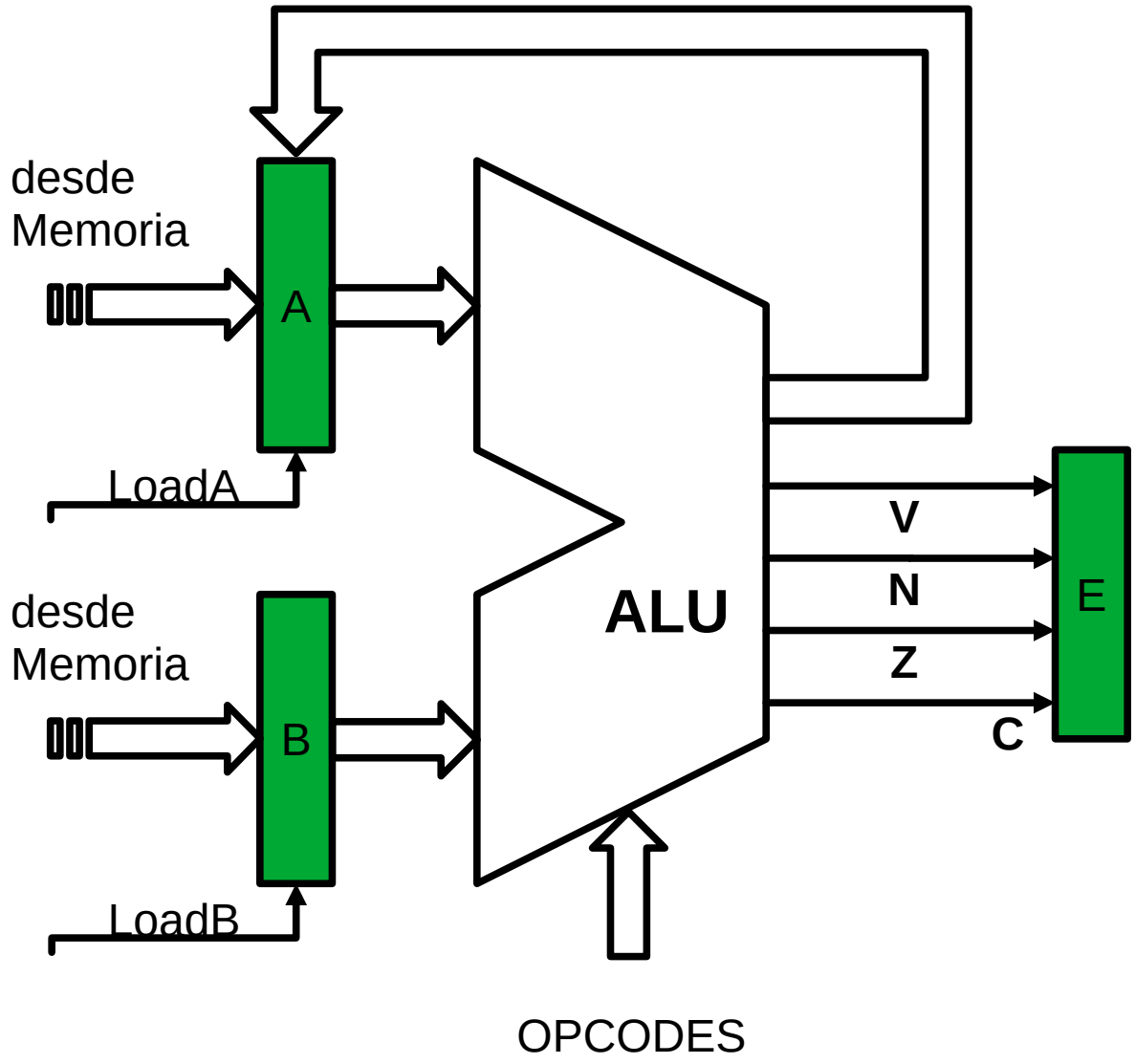

Figura 1: Unidad Aritmético Lógica

El bloque que realiza las operaciones se muestra en la Figura 2 y normalmente se implementa mediante lógica combinatoria.

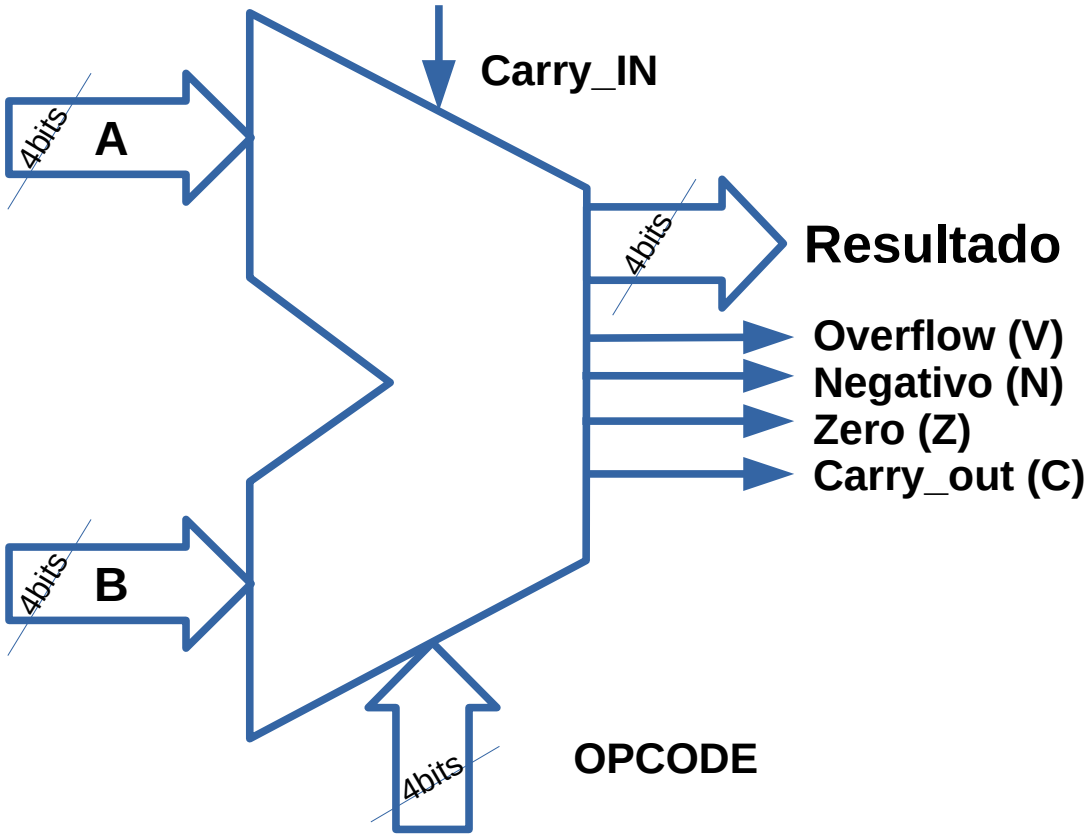

Figura 2: Bloque de Lógica

Descripción del sistema a desarrollar

Debe diseñar una ALU que permita operandos de cuatro (4) bits, con la funcionalidad que se describe a continuación.

Deberá poseer tres registros denominados ***A*** *(Acumulador)*, ***B*** y ***Estado,*** de cuatro (4) bits cada uno.

Los registros ***A*** y ***B*** deberán poseer entradas externas que permitan su carga.

La carga del registro ***A*** se producirá cuando la entrada externa ***LOADA*** se ponga en nivel ALTO.

La carga del registro ***B*** se producirá cuando la entrada externa ***LOADB*** se ponga en nivel ALTO.

El registro ***Estado***, deberá tener cuatro bits (***E_3 E_2 E_1 E_0***) y almacenar los siguientes flags:

- → ***E_3***: *Flag de Overflow* que denominaremos ***V.***
- → ***E_2:*** *Flag Negativo,* que indica que el contenido del ***Acumulador*** representa un número negativo en Complemento a 2, al que denominaremos ***N.***
- → ***E_1***: Flag que indica que el contenido del ***Acumulador*** tiene todos sus bits en BAJO, como esta situación corresponde a la representación de 0, se denominará *Flag Zero* o ***Z.***
- → ***E_0***: *Flag de Acarreo* que indica cuando se produjo un acarreo como resultado de la última operación y que denominaremos ***C.***

Las operaciones que realice la ALU deberán ser las siguientes:

Operación	Código de Operación	Flags que afecta	V	N	Z	C
A = A + B	0000	V, N, Z, C	↕	↕	↕	↕
A = A – B	0001	V, N, Z, C	↕	↕	↕	↕
A = 0xF (SET)	0010	N	R	S	R	R
A = – A	0011	N, Z	●	↕	↕	●
A = 0x0 (CLEAR)	0100	Z	R	R	S	R
A = A+1 (INC A)	0101	V, N, Z, C	↕	↕	↕	↕
NOP ()	0110	-	●	●	●	●
A?B (COMPARAR)	0111	V,N,Z,C	↕	↕	↕	↕
A = LSL (A)	1000	N,Z	●	↕	↕	●
A = LSR (A)	1001	N,Z	●	↕	↕	●
A = A & B (AND)	1010	N,Z	●	↕	↕	●
A = A | B (OR)	1011	N,Z	●	↕	↕	●
A = A ^ B (XOR)	1100	N,Z	●	↕	↕	●
A = ~ A (NOT)	1101	N,Z	●	↕	↕	●
A =ASR(A)	1110	N,Z	●	↕	↕	●
A = B	1111	N,Z	●	↕	↕	●

Referencias:

↕ : El valor puede ser 0 o 1 dependiendo del resultado de la operación

●: El valor no se modifica

R: el valor se establece en 0

S: el valor se establece en 1

- Cuando se realice una operación que involucre un único registro, no deberá afectarse el contenido del otro.

- La operación **A?B (COMPARAR)** realizará la operación **A – B** y deberá dejar ambos registros inalterados, pero deberá establecer los flags.

- La operación **LSL(A)** (Desplazamiento *lógico* a izquierda) desplaza los bits a la izquierda desde el LSB hacia el MSB. El MSB original se pierde y en el LSB se ingresa un 0.

- La operación **LSR(A)** (Desplazamiento *lógico* a derecha) desplaza los bits a la derecha desde el MSB hacia el LSB. El LSB original se pierde y en el MSB se ingresa un 0.

- Las operaciones **AND** , **OR** y **XOR** , se realizan bit a bit entre los registros **A** y **B**.

- La operación **NOT** se realiza invirtiendo todos los bits del registro **A**.

- La operación **ASR(A)** (Desplazamiento *aritmético* a izquierda) desplaza los bits a la derecha desde el MSB hacia el LSB. El LSB original se pierde y se extiende el signo, es decir, se propaga hacia la derecha el MSB (0 si A es positivo o 1 si A es negativo).

- La operación **A = B** reemplaza el contenido del registro A con el contenido del registro **B**.

Tareas a Realizar:

1. Realizar un esquema en bloques de cada operación en donde se vea claramente cuales son los datos de entrada, los bloques que realizan modificaciones sobre los mismos y los datos de salida. A cada uno de los 16 diagramas debe acompañarlo una explicación de como se realizará la operación.
2. Diseñe un decodificador de OPCODES, tenga en cuenta que cada OPCODE, habilitará una operación distinta, por lo que deberá obtener 16 salidas, una para cada bloque, esto es, cuando una salida habilite el bloque correspondiente a la operación, las otras salidas deben deshabilitar al resto de las operaciones
3. A partir de lo obtenido en el punto 1, deberá identificar los bloques que puedan utilizarse en más de una de las operaciones.
    - Si el mismo bloque puede utilizarse en más de una operación, pero con distintos datos de entrada, o produciendo distintas salidas, trate de seleccionar los datos mediante multiplexores o utilice Buffers Tristate con una lógica que maneje Output enable.

- Es conveniente que los registros se implementen con salida TriState y algún tipo de habilitación de escritura. (Input enable).
- Si identifica que para reutilizar un bloque debe agregar logica o registros tambien puede hacerlo.
- Un caso particular es el registro A, piense que debe cargarlo y leerlo en paralelo, pero también deberá desplazar sus bits a izquierda y a derecha. Considere lo visto en la teoría acerca de registros de desplazamiento.

4. A partir de lo que obtenga en el punto anterior deberá decidir la cantidad de bloques minimos necesarios para construir la ALU.
5. A medida que implemente bloques, implemente y simule en Quartus y una vez que esté satisfecho con el diseño del bloque, genere un componente.
6. El diseño final deberá implementarse y simularse en Quartus. Para interconectar los bloques utilice un bus Tri-State.

Ayudas

- Como se vió anteriormente, si se quiere habilitar/deshabilitar el paso de un grupo de señales, se puede utilizar una compuerta **AND** de dos entradas para cada línea: en una de las entradas se conecta la señal deseada y a la otra entrada se aplica un 1 o un 0 según querramos habilitar o deshabilitar el paso de la señal.

- Si se quieren invertir los valores de un grupo de señales, se puede utilizar una compuerta **XOR** de dos entradas para cada línea: en una de las entradas se conecta la señal deseada y a la otra entrada se aplica un 1 o un 0 según querramos invertir o no la señal de entrada.

- Cuando existen múltiples líneas de señal, se pueden agrupar formando **buses**. En Quartus los buses se forman dando el mismo nombre a las líneas que se quieren agrupar y numerándolas consecutivamente a partir de 0.

- En la figura 3, la entrada consta de 4 líneas A0, A1, A2 y A3. Si se desea agruparlas, se escribe A[3..0] el primer número corresponde al bit más significativo y el segundo al LSB. Dentro del Quartus se pueden usar agrupadas o bien individualmente como se muestra en la figura. También se pueden agrupar líneas ya existentes como S1 y S0 en S[1..0].

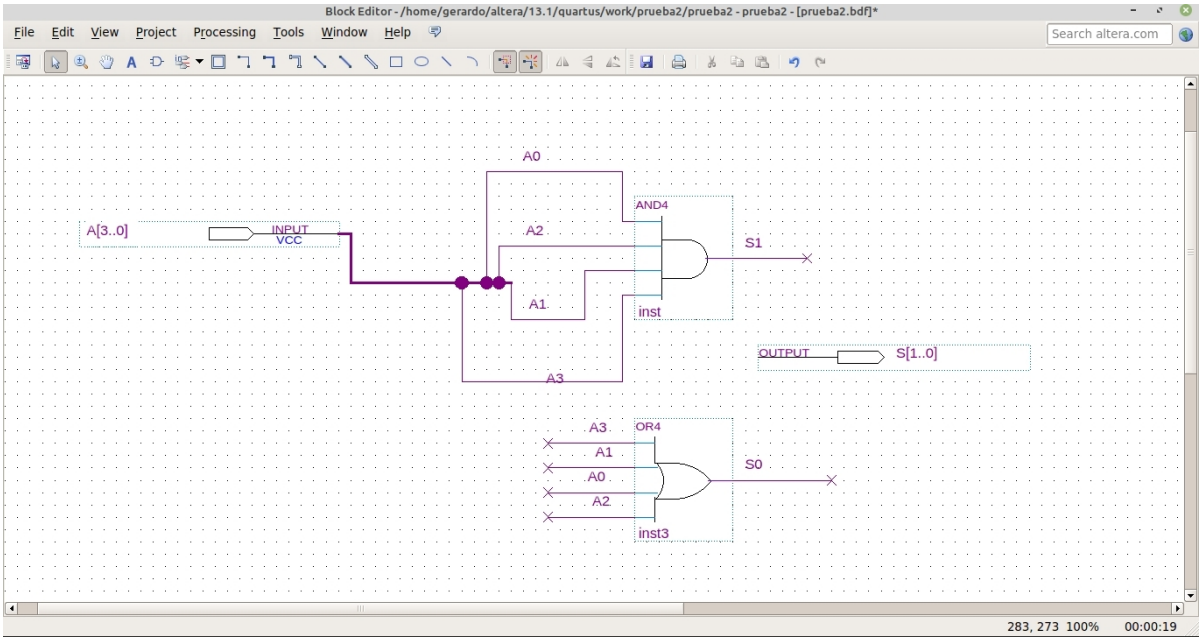

Figura 3

- Se pueden implementar multiplexores que realicen la selección de buses, como se muestra en la figura.

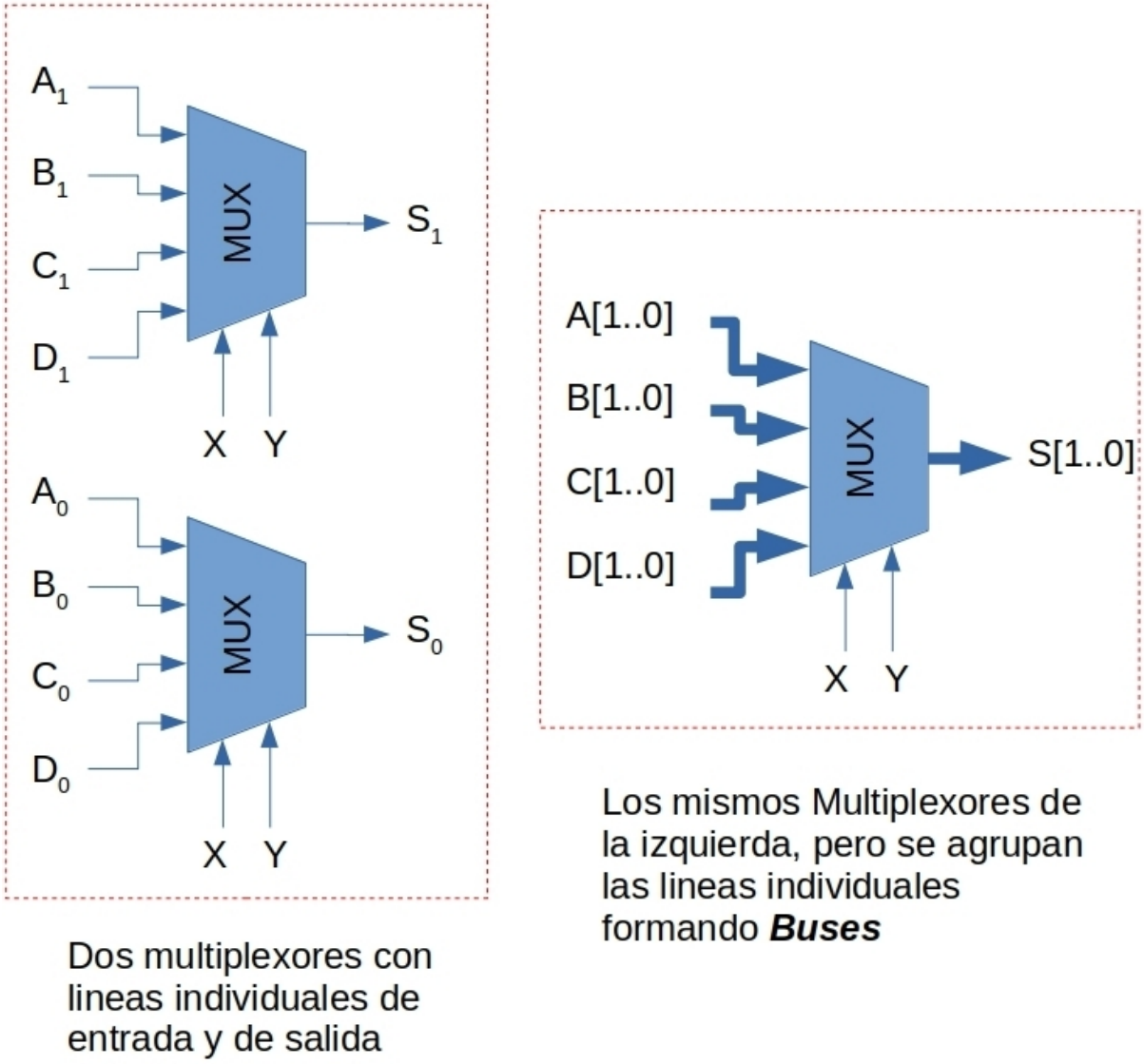

Figura 4

- Se pueden conectar las entradas a distintos bloques, y realizar simultáneamente distintas operaciones, lo importante es seleccionar cual de las SALIDAS se conecta al registro donde debe almacenarse el resultado. Por ejemplo si hay un bloque que realiza la operacion **A AND B** y otro que realiza la operación **A OR B** , ambos pueden tener una conexión entre sus entradas y los registros, pero sus salidas deben conectarse al registro **A**, donde debe almacenarse el resultado, solamente si el **OPCODE** es el correspondiente. (ver figura 5).

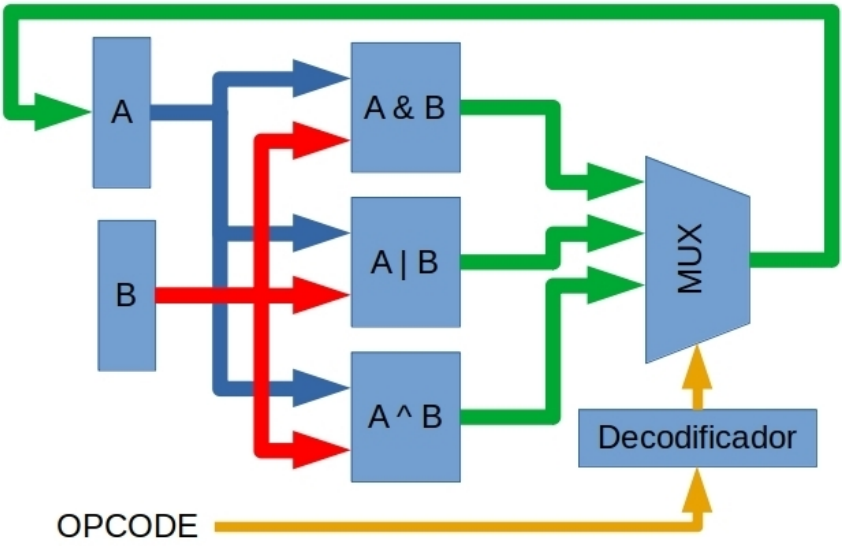

Figura 5

- Alternativamente, las conexiones se pueden realizar utilizando un bus **3-State**, como se muestra en la Figura 6, en este caso el decodificador habilita las salidas de los distintos bloques hacia el registro **A**

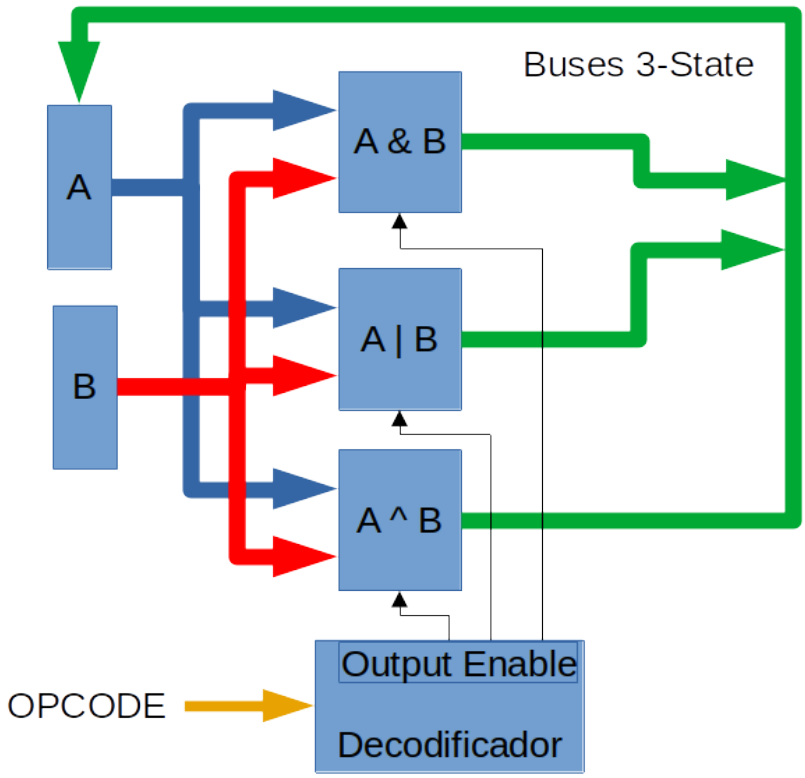

Figura 6