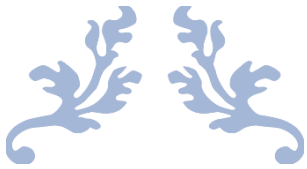


PRÁCTICA COMBINACIONAL

PACO BLAYA COVAS
43231330R

JOAN MARTORELL COLL
43233750Y

1. INDEX

2. INTRODUCCIÓN

Esta práctica consiste en la creación de una Unidad Aritmética-Lógica (ALU) con dos operadores de entrada de 3 bits (A2 A1 A0 i B2 B1 B0) y con 6 operaciones controladas por el código de control C (C2 C1 C0). La salida tiene 3 bits de resultado (R2 R1 R0), además de un bit extra para casos especiales (E). La siguiente tabla define cada una de las operaciones de la ALU:

C2 C1 C0	Operación por realizar	Aclaramientos
0 0 0	Transformación de SM a C2	La salida R es la representación en C2 del valor representado en SM en la entrada A. E = 1 si A no es representable en C2, E = 0 en el otro caso.
0 0 1	Transformación de C2 a SM	La salida R es la representación en SM del valor representado en C2 en la entrada A. E = 1 si A no es representable en SM, E = 0 en el otro caso.
0 1 0	A − B	Los dos valores por operar están representados en C2. E = 1 si se produce overflow, E = 0 en el otro cas.
0 1 1	Comparador	Los dos valores por operar están representados en C2. R = 001 si A > B. R = 010 si A = B. R = 100 si A < B.
1 0 0	A × 2	El resultado y A están representados en BNSS. El bit E se utilizará como si fuese el cuarto bit de la salida, es decir, R3.
1 0 1	Número primo	R = 111 si A es un número primo, donde A está representado en BNSS. R = 000 en el otro caso.

3. DESCRIPCIÓN DE LAS PARTES IDENTIFICADAS

3.1 ALU:

La ALU consiste en un archivo principal llamado *solucio.dig*, donde solo aparecen les entradas (A2 A1 A0 y B2 B1 B0), las entradas de selección (C2 C1 C3), las salidas (R2 R1 R0 y E) y un componente personalizado guardado en otro archivo llamado *alu.dig*.

Este componente (*alu.dig*) se encarga de unir todos los subprogramas (000, 001, 010, ...); es decir, a partir de un decodificador (proporcionado por el Digital) de tres entradas de selección (C2 C1 C0), se selecciona el subprograma que se desea utilizar. Luego, todas las salidas R2 se juntan en una puerta OR, para que, solo con una activa, la salida final se active; y así para todas las salidas (R2 R1 R0 y E).

Cada uno de estos subprogramas está conectado a las entrades necesarias (algunos a solo la entrada A, y otros a A y B), al igual que las salidas (algunos solo la salida R i otros también utilizan la E). Pero lo que tienen todos los subprogramas en común es una entrada extra llamada EN; si esta es 0, todas las salidas del subprograma serán 0; i si es 1, el subprograma funcionará según haya sido creado. Esta operación se lleva a cabo con puertas AND que conectan el último cable de cada salida con la entrada EN.

A continuación, se explicará cada subprograma. Cabe decir que a la hora de sacar la tabla de verdad i las funciones de cada salida no se ha tenido en cuenta la entrada EN, ya que la hemos implementado en el momento de hacer el circuito en Digital; es decir, hemos creado el circuito sin tener en mente la entrada EN y la hemos incorporado al final conectándola con puertas AND a los cables de salida.

3.2 SM a C2 (000):

Para pasar de Signo y Magnitud a Complemento a 2 podemos empezar sacando la tabla de verdad: cuando A2 valga 0 se hará referencia a los números positivos (0,1,2,3) y cuando valga 1 serán los números negativos (-0,-1,-2,-3). Como vemos, en ambos casos el cero está incluido, esto se debe a que en Signo y Magnitud se puede representar de dos formas: 000 y 100. Teniendo esto en cuenta podemos saber que la salida extra E siempre será 0 ya que se pueden representar todos los valores de Signo y Magnitud.

Tabla de verdad						
Entradas			Salidas			
A2	A1	A0	E	R2	R1	R0
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

De esta operación las funciones mínimas quedarían tal que así:

- E = 0
- R2 = A2·A1+A2·A0
- R1 = A2'·A1 + A1·A0' + A2·A1'·A0
- R0 = A0

Ahora solo nos ha hecho falta crear el subprograma en el Digital con la ayuda de las funciones, incorporar la entrada EN al final i añadirlo como componente personalizado en el archivo *alu.dig*.

3.3 C2 a SM (001):

Para la conversión de Complemento a 2 a Signo y Magnitud, primero sacaremos la tabla de verdad, como hemos hecho anteriormente: cada número positivo en C2 (0XX) es el mismo que en SM (0XX); menos el 0 (000_{C2}), el cual convertido a SM puede ser 100_{SM} o 000_{SM}, por eso ponemos una X. Por otro lado, para convertir los números negativos (1XX), los hemos pasado a decimal para luego convertirlos a SM. No obstante, nos ha surgido un problema en el 100_{C2}: si lo convertimos a SM es 1100, como está compuesto por cuatro bits no es representable en SM (máximo 3 bits), se activaría la salida E y la salida R sería indiferente (X).

Tabla de verdad

Entradas			Salidas			
A2	A1	A0	E	R2	R1	R0
0	0	0	0	X	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	1	X	X	X
1	0	1	0	1	1	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

Con el uso de la tabla de verdad, hemos sacado las funciones mínimas:

- E = A2·A1'·A0'
- R2 = A2
- R1 = A1'·A2 + A0'·A1 + A1·A2'
- R0 = A0

Ahora solo nos ha hecho falta crear el subprograma en el Digital con la ayuda de las funciones, incorporar la entrada EN al final i añadirlo como componente personalizado en el archivo *alu.dig*.

3.4 A - B (010):

Para restar A entre B, nos hemos tenido que dar una vuelta por la teoría ya que no es un circuito que se vea a simple vista. Para restar en Complemento a 2 debemos cambiar de signo al segundo operando, en este caso B, y sumarlo al primer operando, A. Para hacer el cambio de signo, hemos negado todas las entradas y le hemos sumado 1. Ahora tenemos A, y B con signo cambiado, los cuales debemos sumar entre ellos. Luego, conectamos las salidas del sumador a las salidas R.

Tabla de verdad				
Entradas				Salida
I	A	B	R	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Para la salida E, donde hay overflow, nos ha sido más complicado: necesitábamos saber el bit más representativo de A, B (signo cambiado) y R. Establezcamos que A = bit más representativo de A, B = bit más representativo de B y R = bit más representativo de R. Si A y B son diferentes nunca hay overflow, y si A y B són iguales debemos ver cómo se comporta R. No obstante, en caso de hacerlo con solo estos datos, nos va a salir que hay overflow cuando los dos números sean iguales, cosa que no debería suceder. Para solucionarlo, compararemos los números con un comparador del Digital para saber cuándo son iguales (I), en tal caso, no habrá overflow. Con estos datos, haremos la tabla de verdad y sacaremos la función mínima de E:

- $E = I' \cdot A' \cdot B' \cdot R + I' \cdot A \cdot B \cdot R'$

Ahora sólo nos falta incorporar esta función y añadir la entrada EN al final.

3.5 Comparador (011):

Para hacer el comparador, utilizaremos el comparador del Digital. Uniremos todas las entradas a este y le activaremos la casilla de “Operación con signo”. Por otro lado, el circuito funciona correctamente siempre y cuando las dos entradas no estén negativo a la vez; porque cuando lo están, los resultados nos salen al revés (A<B cuando A>B y viceversa). Para solucionarlo, lo detectaremos con una puerta AND conectada a los bits más significativos, esta puerta valdrá 1 cuándo los dos números estén en negativo, que a su vez pasará por un decodificador de una entrada de selección: cuando la primera entrada esté activada, hará que las salidas del comparador vayan a sus respectivas salidas según el enunciado; sin embargo, cuando la segunda salida del decodificador esté activada, hará que las salidas del comparador vayan a la salida contraria del enunciado. Para finalizar, incorporaremos la entrada EN.

3.6 A x 2 (100):

Para hacer este apartado, en lugar de tener que usar un multiplicador, sabiendo que es lo mismo multiplicar A*2 que sumar A dos veces, hemos construido un full adder con la misma entrada. Es decir, como A esta multiplicada por un numero par nunca nos saldrá como resultado un numero impar.

Para la creación de esta parte, hemos hecho uso de 3 full-adders junto a una única entrada A (A0, A1 y A2) con sus respectivas salidas R0, R1, R2 y una salida extra R3 o también nombrada E para el caso que la multiplicación supere los 3 bits, (a partir de A=4). Por otro lado, en el primer full-adder nos piden un carry-in, pero como el primer sumador no tiene hemos establecido un valor fijo de 0 como podremos ver en el siguiente apartado de circuito de las partes identificativas.
Así, la tabla de verdad quedaría de esta forma:

Tabla de verdad						
Entradas			Salidas			
A2	A1	A0	E/R3	R2	R1	R0
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	1	1	0
1	0	0	1	0	0	0
1	0	1	1	0	1	0
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Y las funciones mínimas tal que así:

- E/R3 = A2
- R2 = A1
- R1 = A0
- R0 = 0

Creamos el circuito en el Digital y no nos olvidamos de incorporar la entrada EN al final.

3.7 Número primo (101):

En esta parte, donde debemos comprobar si la entrada A (representado en BNSS) es un numero primo, primero hemos sacado la tabla de verdad: sabemos que los primeros números primos son 2, 3, 5y 7; es decir, 010_{BNSS}, 011_{BNSS}, 101_{BNSS} y 111_{BNSS}. Entonces, solo debemos poner un uno donde es primo.

Tabla de verdad			
Entradas			Salida
A2	A1	A0	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

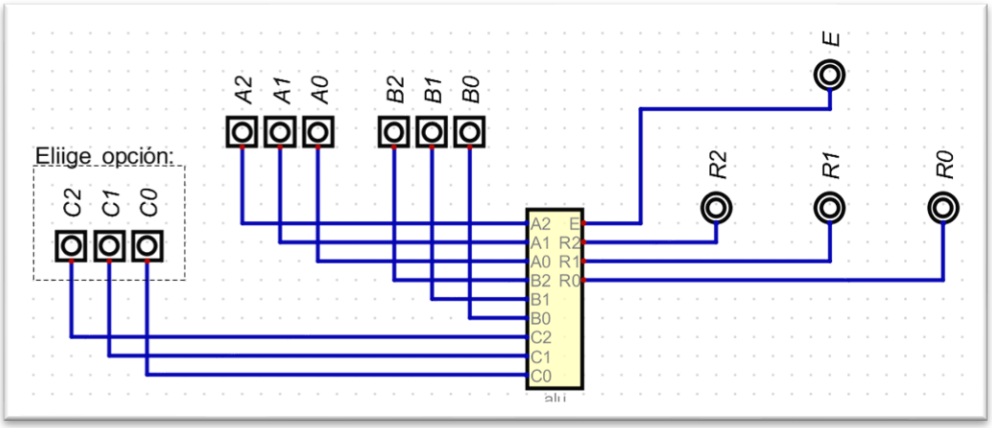
A continuación, sacamos la función de la salida:

- $S = A0 \cdot A2 + A1 \cdot A2'$

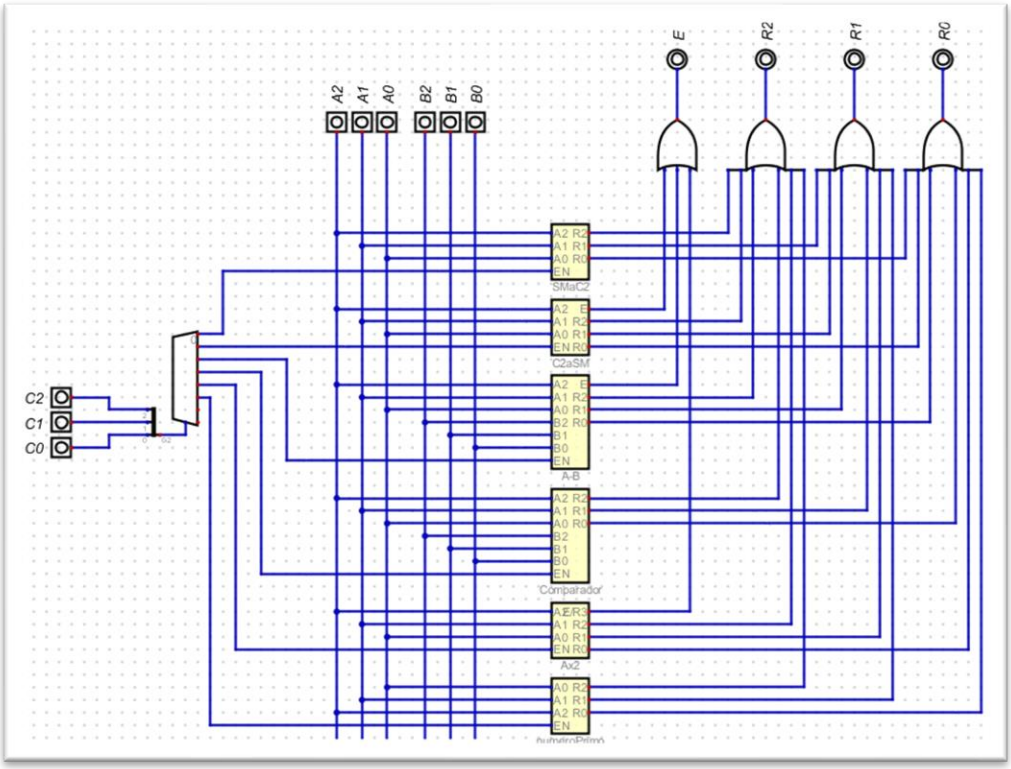
Finalmente, creamos el circuito en el Digital, añadimos la entrada EN al final y conectamos la única salida a todas las salidas, ya que si es primo todos los dígitos de R deben ser 1 y si no es primo deben ser 0.

4 CIRCUITO DE LAS PARTES IDENTIFICADAS

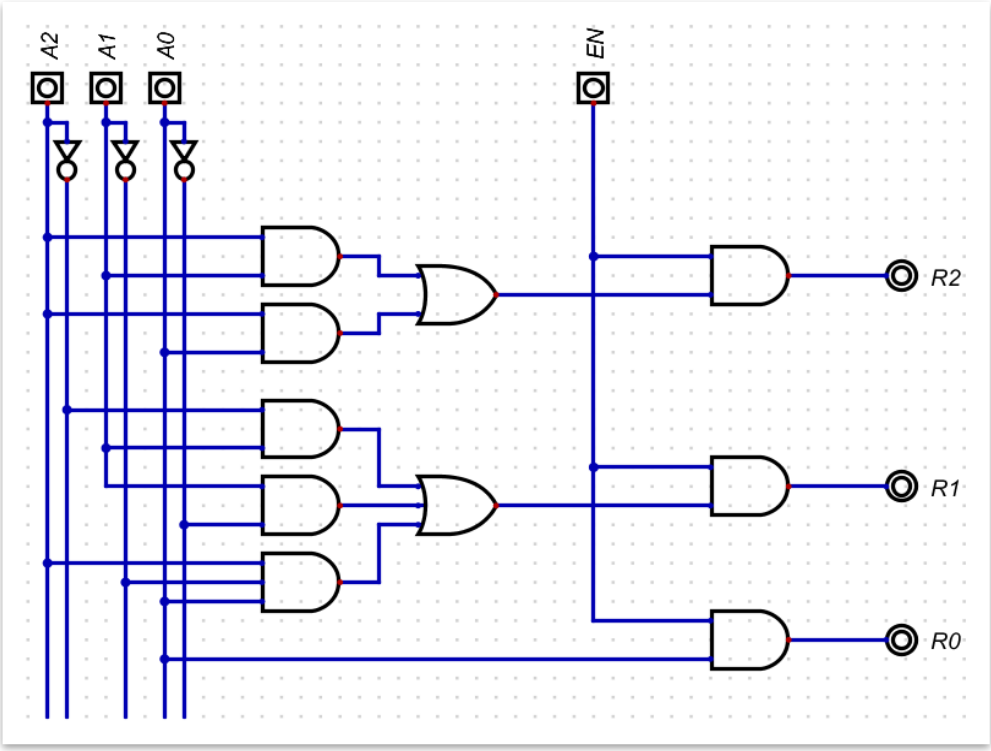
4.1 Solucion:



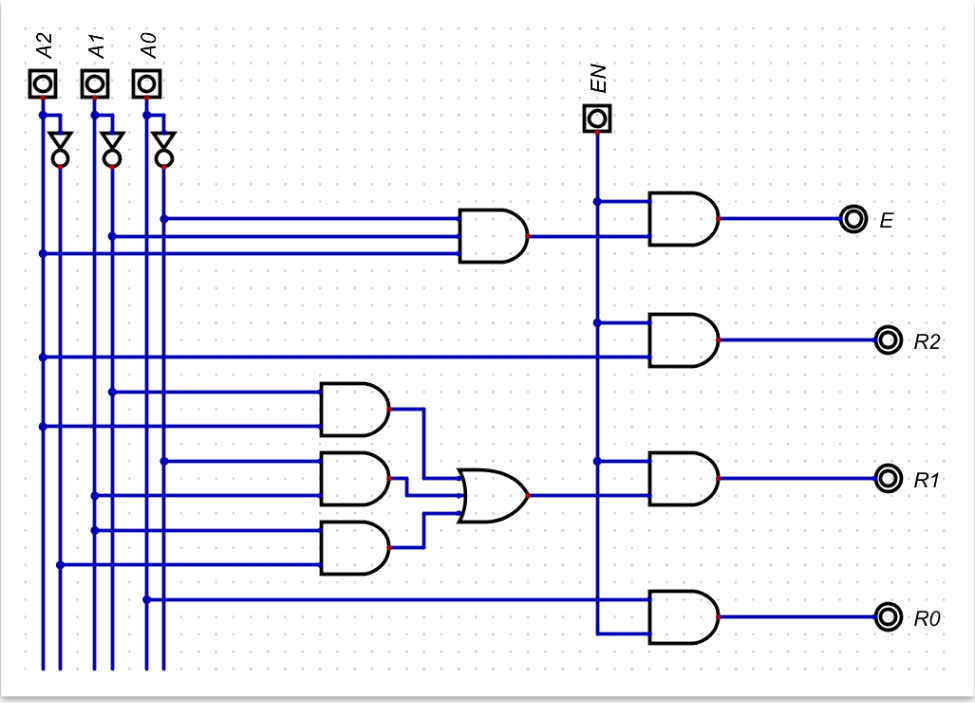
4.2 ALU:



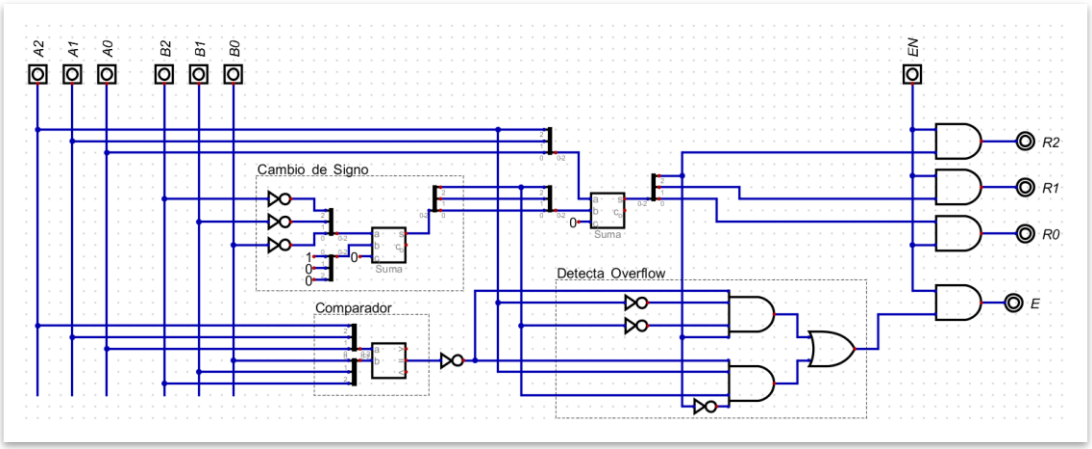
4.3 SM a C2 (000):



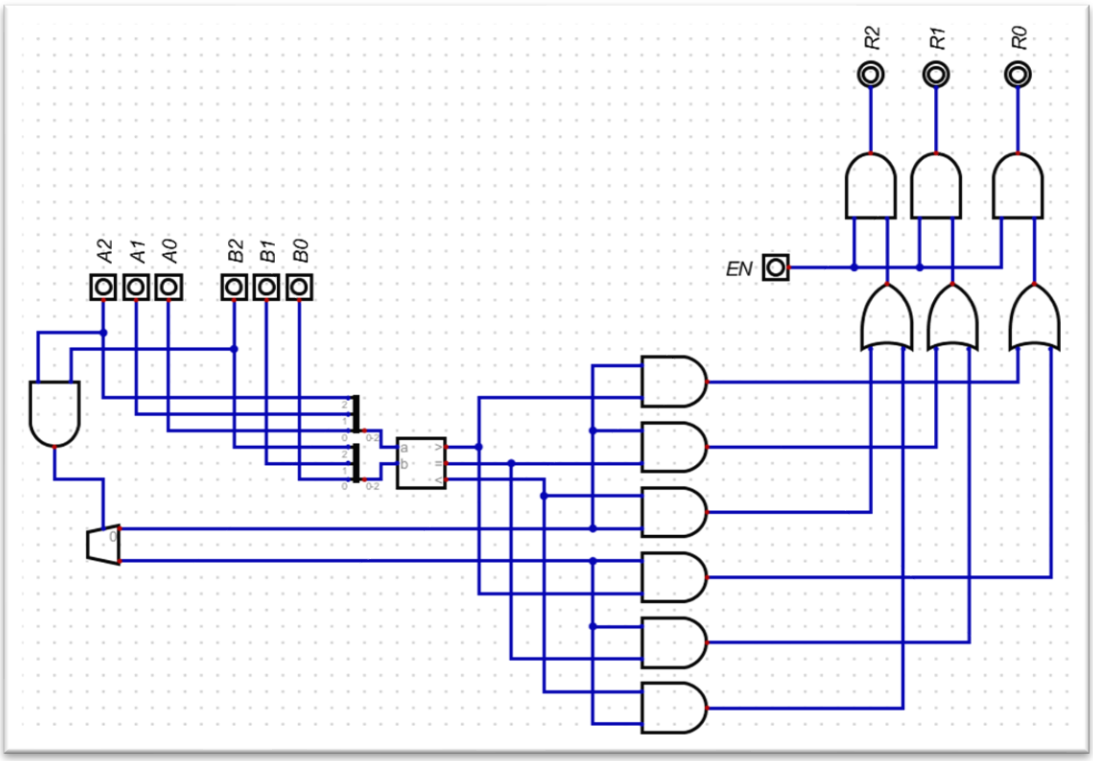
4.4 C2 a SM (001):



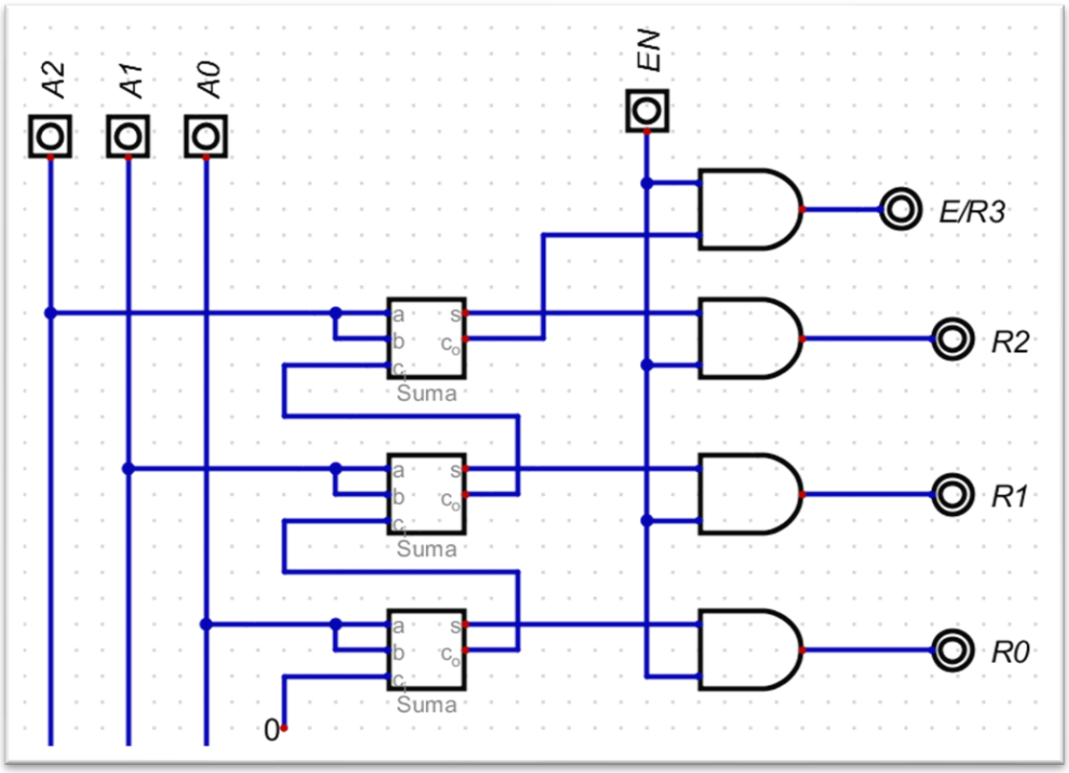
4.5 A – B (010):



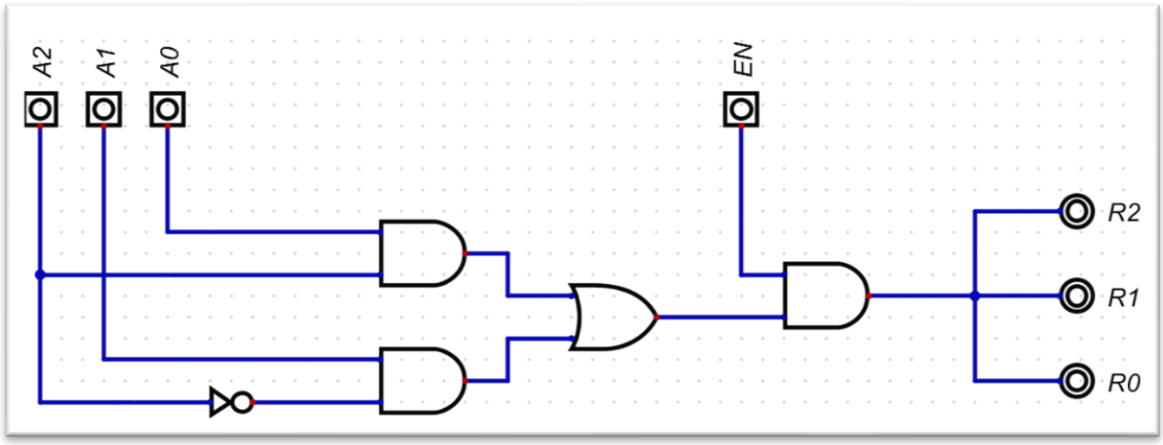
4.6 Comparador (011):



4.7 A x 2 (100):



4.8 Número primo (101):

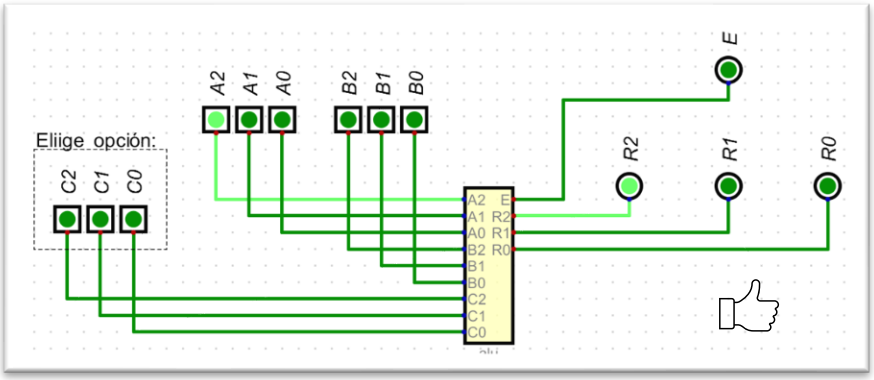


5 JUEGO DE PRUEBAS

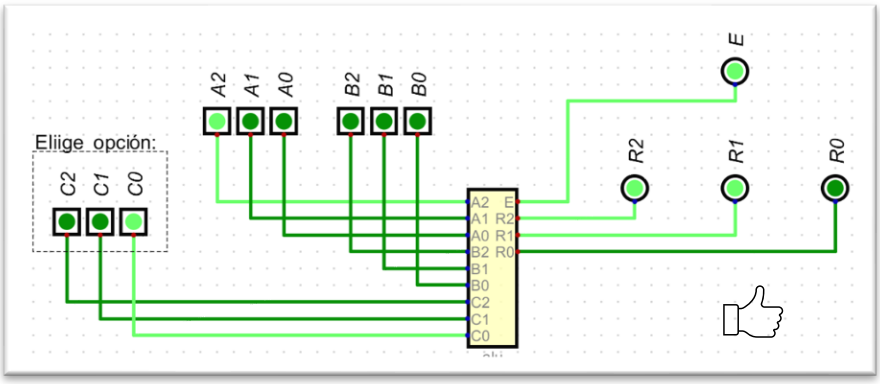
5.1 ALU:

Para comprobar que la ALU funciona correctamente, pondremos a prueba las seis diferentes operaciones que se pueden realizar con los mismos dígitos para que sea más fácil ver la diferencia:

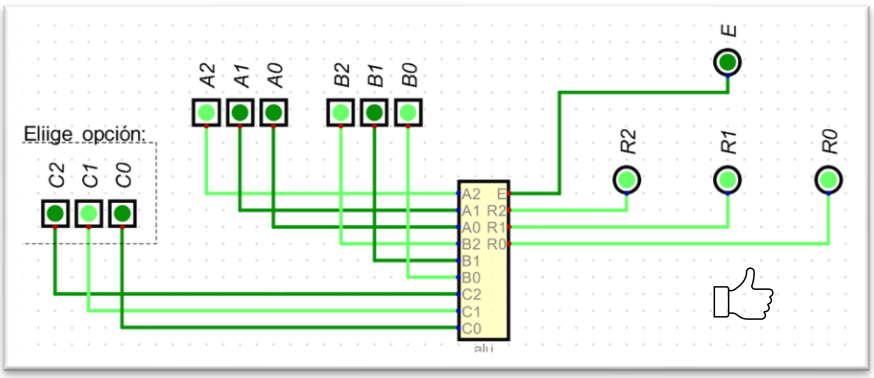
- SM a C2 => Entrada A: 100. Salida R: 100



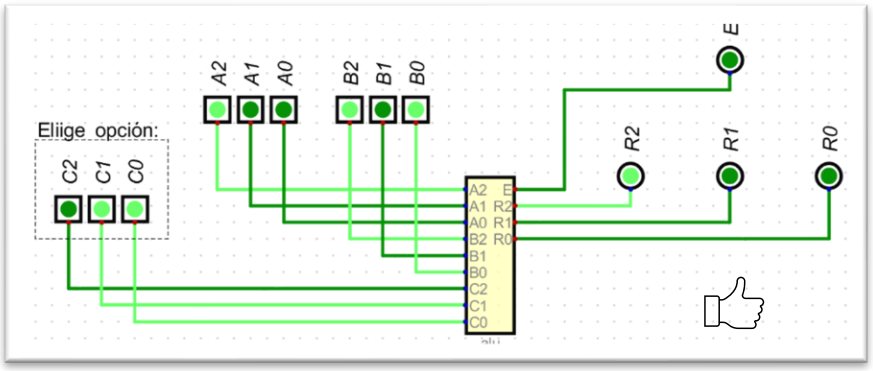
- C2 a SM => Entrada A: 100. Salida R: 110, E: 1



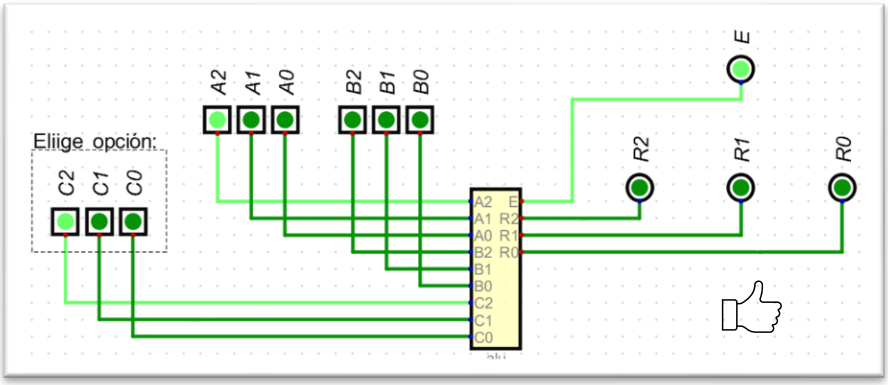
- A – B => Entrada A: 100, B: 101. Salida R: 111, E:0



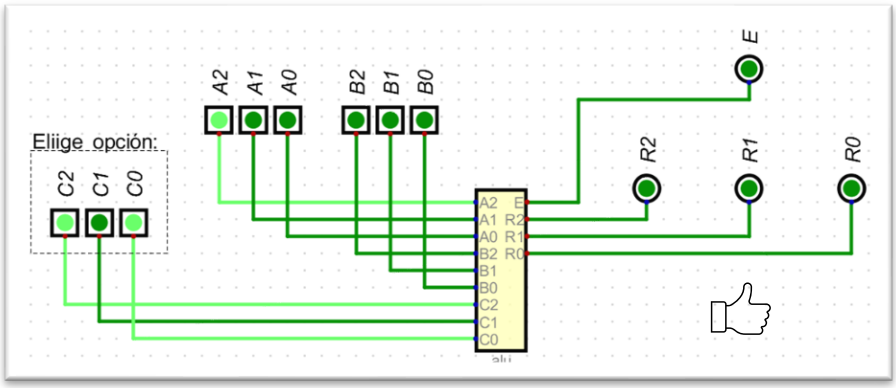
- Comparador => Entrada A: 100, B: 101. Salida R: 100



- A x 2 => Entrada A: 100. Salida R: 000, E: 1



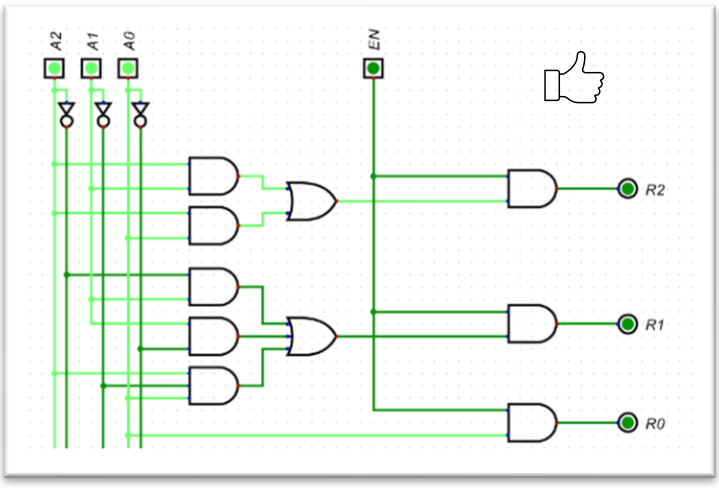
- Número primo => Entrada A: 100. Salida R: 000



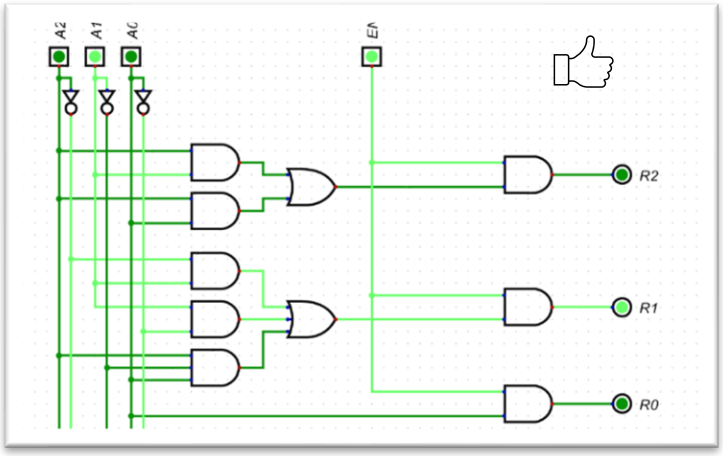
5.2 SM a C2 (000):

A la hora de verificar que el circuito va como debería hemos comprobado con varias combinaciones, entre ellas con la salida EN desactivada, con un numero negativo y con otro positivo; y, como hemos podido comprobar, funciona correctamente:

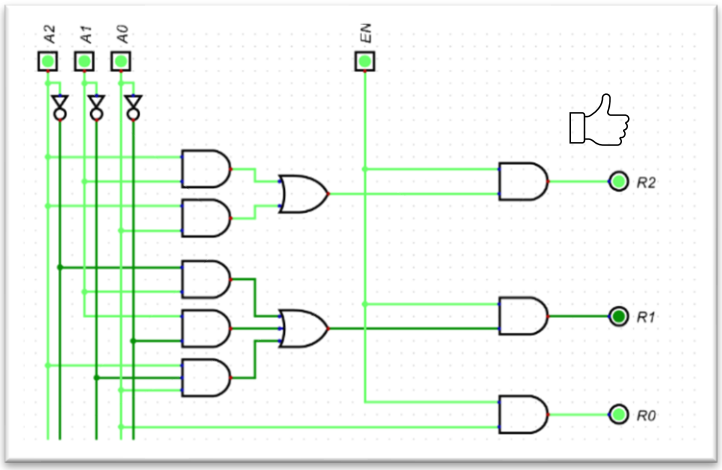
- En el caso de EN desactivado, el circuito no responde.



- Cuando ponemos un número positivo, nos devuelve el mismo.



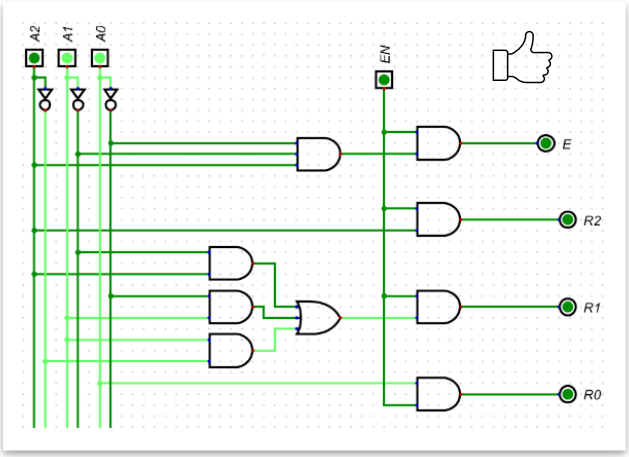
- Con un número negativo, nos lo transforma correctamente.



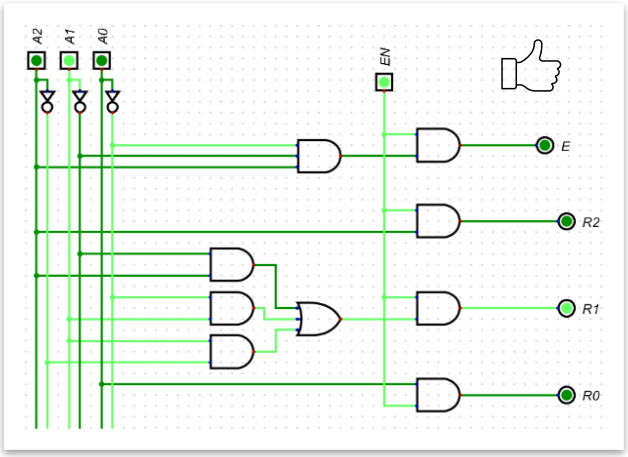
5.3 C2 a SM (001):

Para comprobar que el circuito funciona correctamente, haremos la prueba con cuatro combinaciones posibles:

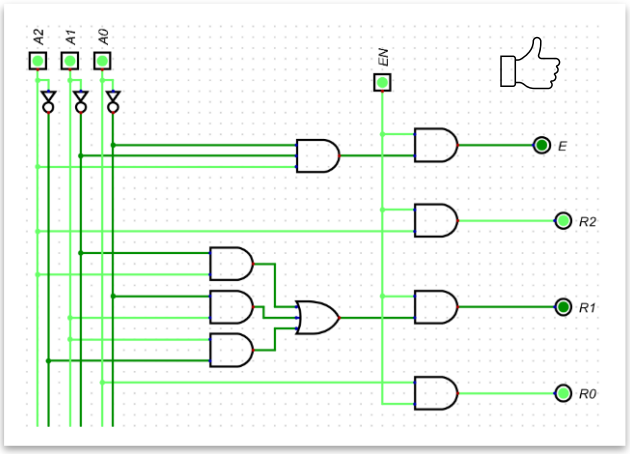
- Entrada EN desactivada, el circuito no responde



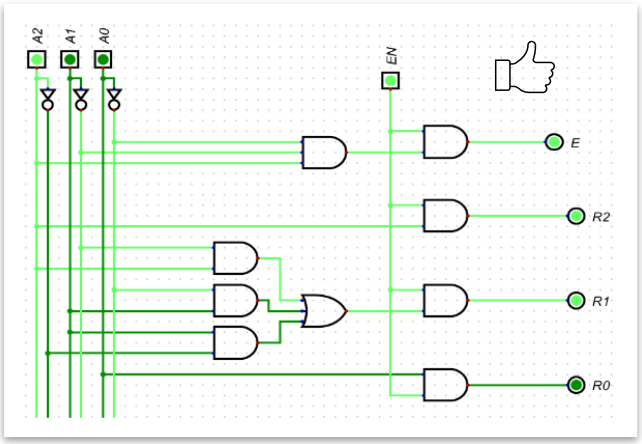
- Entrada A en positivo, en SM será el mismo



- Entrada A en negativo, transforma al número correspondiente



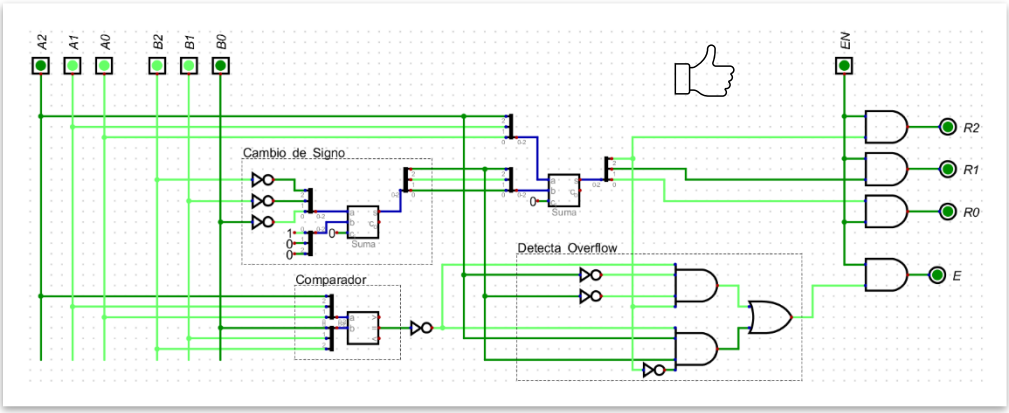
- Entrada A = 100, la salida E debe dar 1.



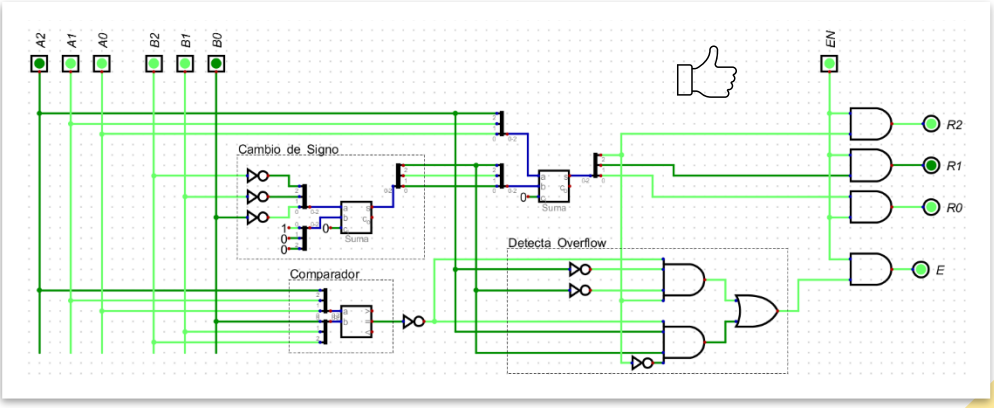
5.4 A – B (010):

Verificaremos que el circuito funciona correctamente probando cinco combinaciones distintas:

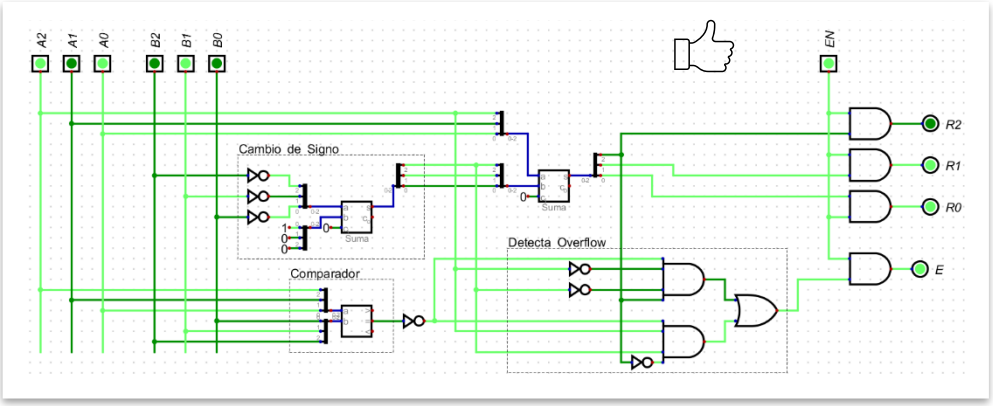
- Entrada EN desactivada, el circuito no responde



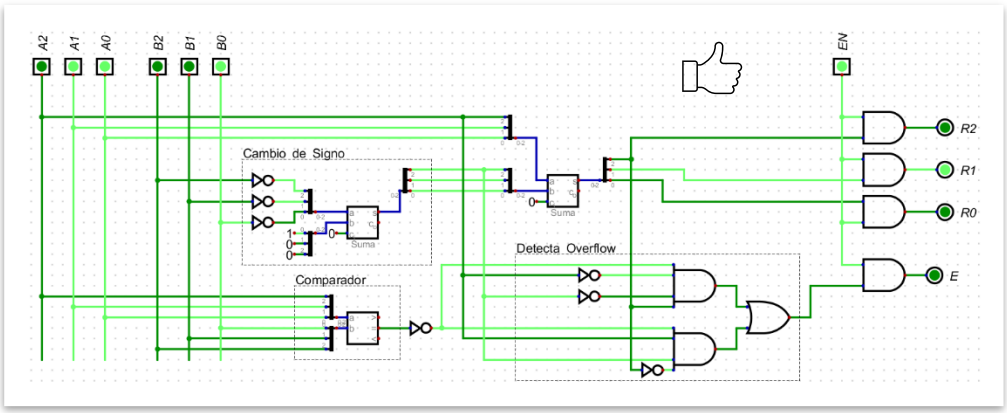
- Número positivo menos número negativo, resultado debe ser positivo. En este caso como da 4, no es representable en C2 y hay overflow



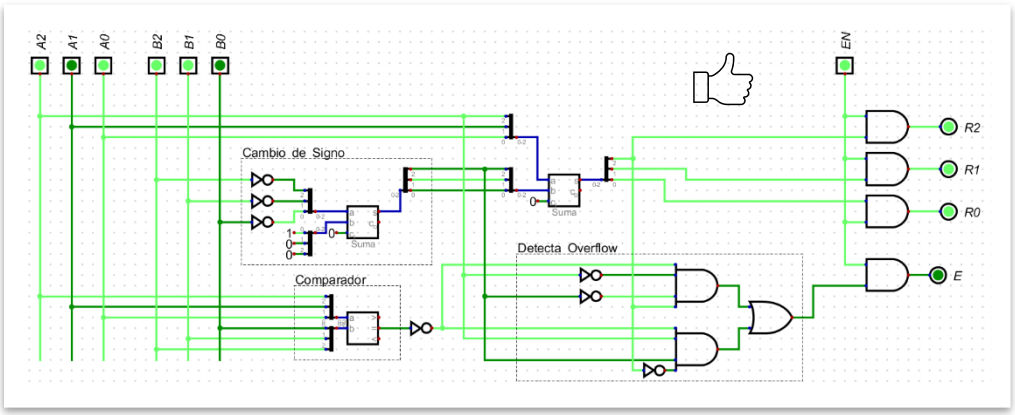
- Numero negativo menos número positivo, resultado debe ser negativo. En este caso como da -5, no es representable en C2 y hay overflow



- Número positivo menos número positivo, resultado correspondiente



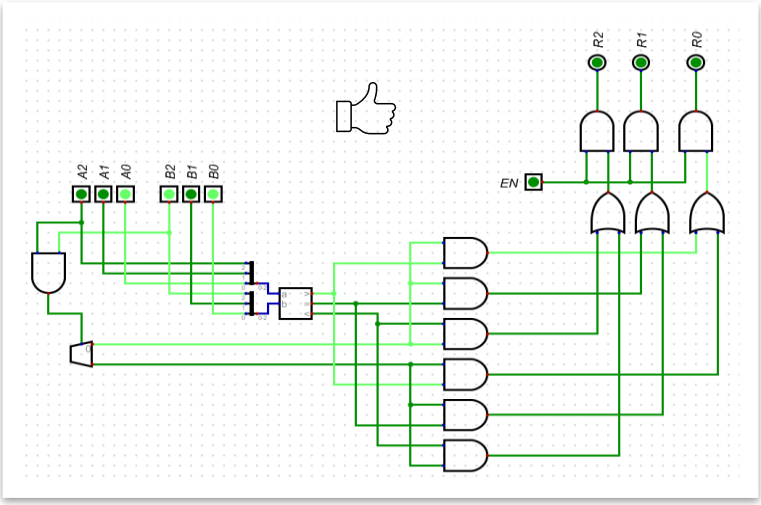
- Número negativo menos número negativo, resultado correspondiente



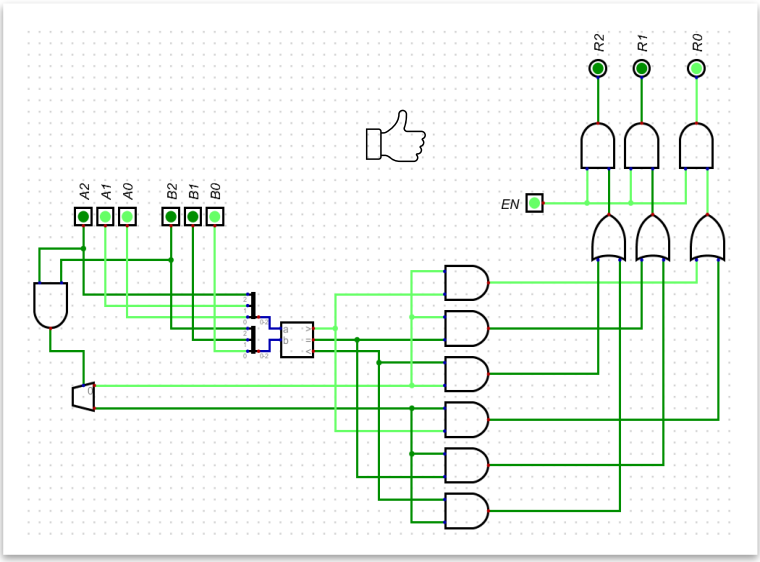
5.5 Comparador (011):

Para comprobar que el comparador funciona correctamente, probaremos cuatro combinaciones diferentes:

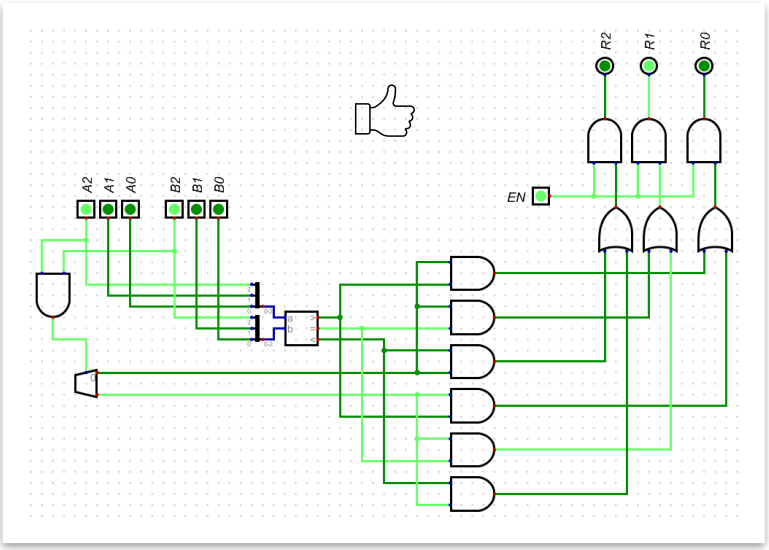
- Entrada EN desactivada, el circuito no responde



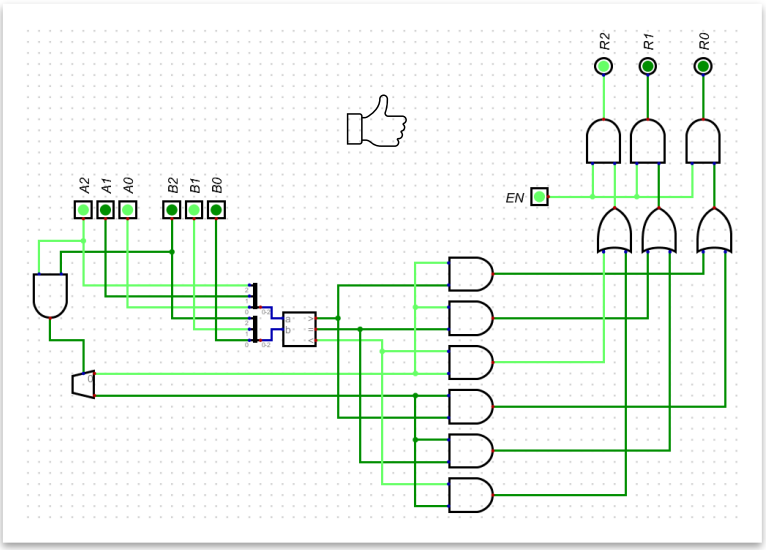
- Entrada A más grande que entrada B



- Entrada A igual que entrada B

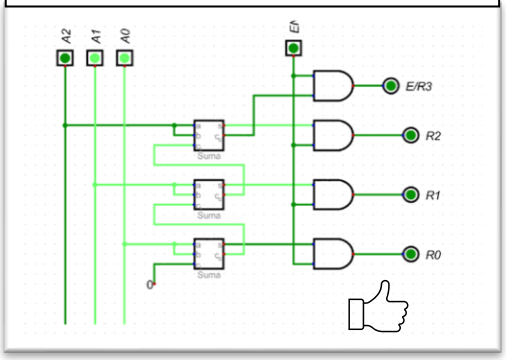
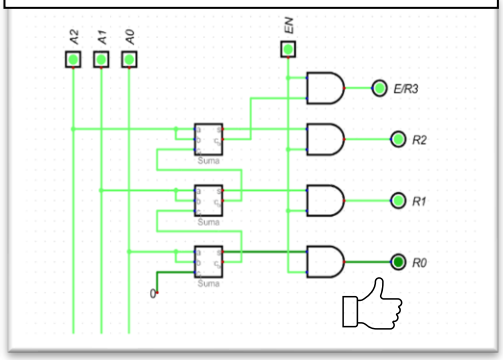


- Entrada B más grande que entrada A



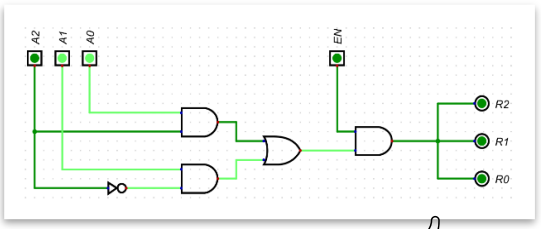
5.6 A x 2 (100):

En este circuito hemos puesto como juego de pruebas dos casos claves: cuando EN esta activado y desactivado y, como vemos el circuito funciona sin problemas (nos podemos fijar que al multiplicar por 2, la salida R0 nunca se encenderá por que indica los números impares)

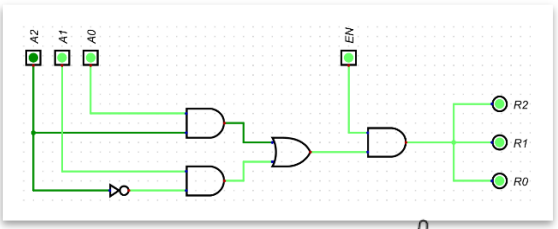
EN desactivado	EN activado
	

5.7 Número primo (101):

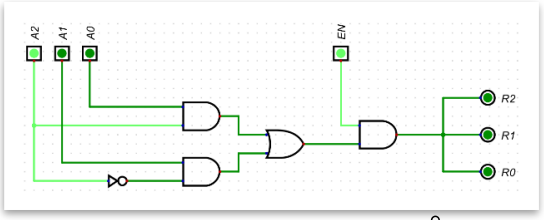
Para verificar que el circuito funciona correctamente, lo haremos mostrando el resultado cuando la entrada EN esté desactivada, y probaremos algunos números en BNSS para ver si son primos o no.



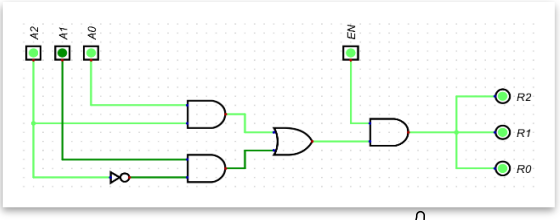
Entrada EN desactivada



Número 3 es primo



Número 4 no es primo



Número 5 es primo

6 CONCLUSIONES

En conclusión, creemos que esta práctica ha sido un conjunto de problemas a los que les hemos tenido que ir poniendo una solución; es decir, cualquier persona que necesite pasar de C2 a SM podrá hacerlo fácilmente con nuestra ALU, sin tener que hacerlo manualmente (que se tarda mucho más tiempo), y así para las diversas operaciones que hemos hecho. Que es básicamente de lo que trata un ordenador, con unas entradas propuestas por el usuario y un código de selección, el ordenador te dará la respuesta deseada.

En general, la mayoría de las operaciones las hemos hecho sin problemas; sin embargo, en el apartado A – B, se nos ha complicado para detectar el overflow, pero finalmente lo hemos solucionado.

Además, esta práctica nos ha ayudado ha fortalecer nuestros conocimientos sobre las puertas lógicas, los diferentes componentes lógicos y las conversiones entre los distintos tipos de binario. Estamos seguros de que más adelante lo vamos a necesitar más que nunca.