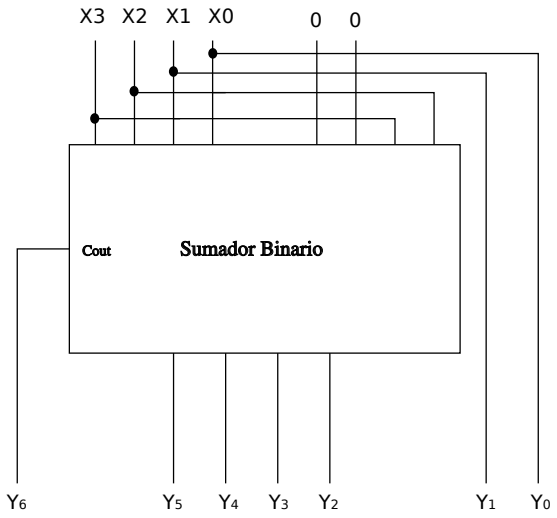
FUNDAMENTOS DE LOS COMPUTADORES
Ejercicios Resueltos Tema 2: Sistemas Combinacionales

1. **Diseña un circuito que acepte como entrada un número sin signo de 4 bits y que genere como salida el producto de dicho número por 5, teniendo en cuenta que realizar el producto de un número binario por 2^i equivale a añadir a ese número i ceros a su derecha. Idem pero multiplicando por 6.**

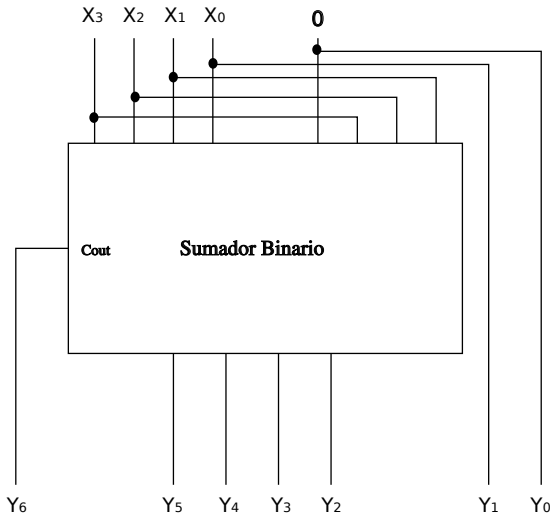
a) $Y = 5 \times X = 4 \times X + X$

$$\begin{array}{rr} 4 \times X & X_3 X_2 X_1 X_0 \;\; 0 \;\; 0 \\ X & + \;\; X_3 X_2 X_1 X_0 \\ \hline & Y_6 \; Y_5 Y_4 Y_3 \; Y_2 \; Y_1 \; Y_0 \end{array}$$



b) $Y = 6 \times X = 4 \times X + 2 \times X$

$$\begin{array}{rr} 4 \times X & X_3 X_2 X_1 X_0 \;\; 0 \;\; 0 \\ 2 \times X & + \;\; X_3 X_2 X_1 X_0 \;\; 0 \\ \hline & Y_6 \; Y_5 Y_4 Y_3 \; Y_2 \; Y_1 \; Y_0 \end{array}$$



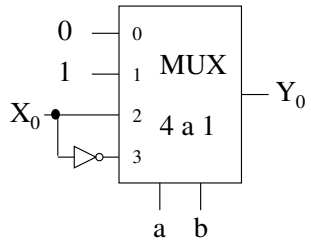
2. **Construye utilizando muxes y puertas lógicas el dispositivo definido mediante la siguiente tabla, donde X_0, X_1, X_2 y X_3 son entradas dato, a y b son entradas de control e Y_0, Y_1, Y_2 e Y_3 son salidas.**

a	b	Y_0	Y_1	Y_2	Y_3
0	**0**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**1**	**1**
1	**0**	X_0	X_1	X_2	X_3
1	**1**	$\overline{X}_0$	$\overline{X}_1$	$\overline{X}_2$	$\overline{X}_3$

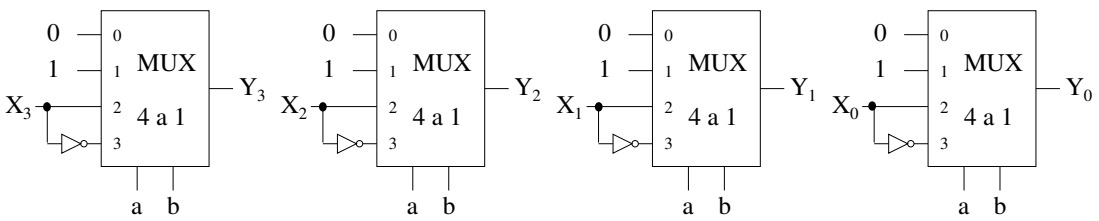
En primer lugar, debemos tener en cuenta que la solución consiste en 4 funciones lógicas independientes, aunque muy parecidas entre sí. Nos centraremos en implementar Y_0:

a	b	Y_0
0	0	0
0	1	1
1	0	X_0
1	1	$\overline{X}_0$

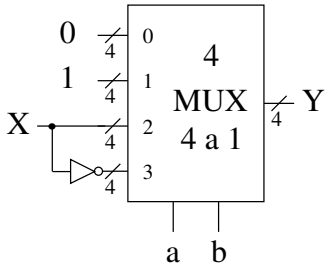
Tal y como se ve en la tabla, la función Y_0 puede tener hasta 4 posibilidades: 0, 1, X_0 y $\overline{X}_0$. Para saber cuál va a tomar necesitamos conocer el valor de las variables de control a y b. En la teoría hemos visto que hay un circuito que se adapta a este esquema: el *multiplexo* (MUX), que selecciona la salida entre varias entradas. En nuestro caso, Y_0 puede variar entre 4 valores, por lo tanto necesitaremos un *MUX 4 a 1*:



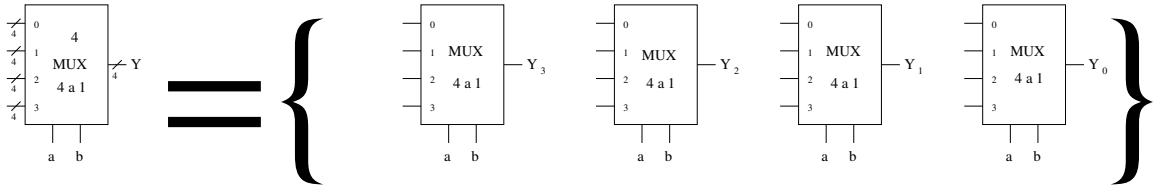
Como nos piden implementar 4 bits de salida la solución pasa por replicar el mismo módulo y adaptarlo a cada función:



Podemos introducir una nueva representación que nos permita simplificar y generalizar los diseños. Para ello vamos a orientar la representaciones a *buses* de datos y variables de n bits en vez de líneas y variables de un único bit. La solución sería entonces:



Donde se utiliza la representación:



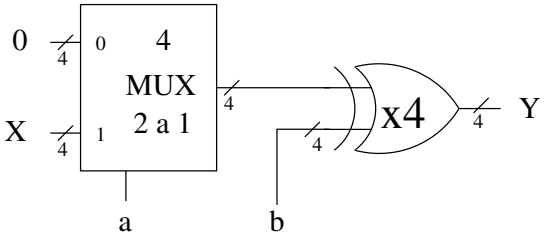
SOLUCIONES ALTERNATIVAS

La solución comentada es válida para cualquier situación en donde la salida pueda tomar diferentes valores independientes entre si. De todos modos, si las diferentes operaciones están relacionadas, entonces se podrían agrupar.

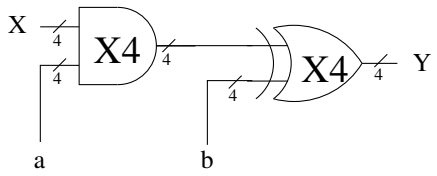
En nuestro caso, recordando las operaciones necesarias para cada bit de salida:

a	b	Y_i
0	0	0
0	1	1
1	0	X_i
1	1	$\overline{X}_i$

Podemos notar que las operaciones impares (1 y 3) son las negadas de las pares (0 y 2, respectivamente). Tal y como hemos visto en teoría en el sumador/restador (sección 2.2.6), esa operación de mantener el valor o negarlo se puede realizar con una puerta EXOR. Por lo tanto, en el circuito final sólo sería necesario utilizar un *MUX 2 a 1* y puertas EXOR a la salida:



Más aún, las dos operaciones que quedan (elegir entre 0 o X_i) también se pueden implementar con una única puerta lógica, una puerta AND:

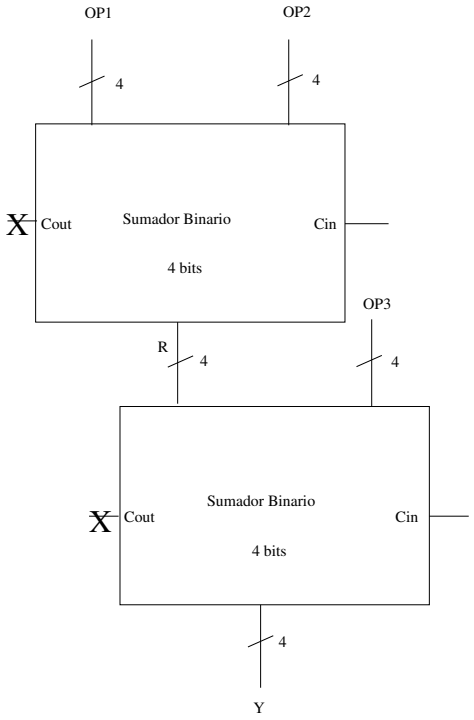


3. **Siendo A, B, C e Y enteros de 4 bits en complemento a 2, diseña un dispositivo que implemente el siguiente algoritmo:**

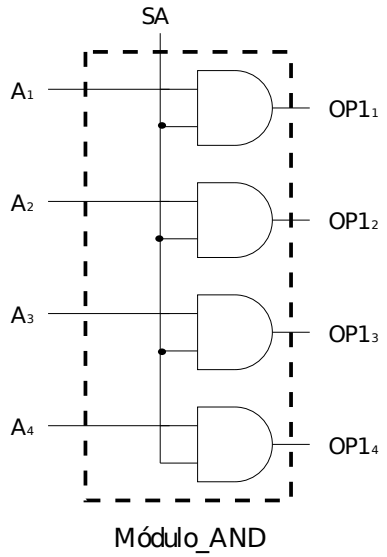
```
If A > B then
     If A > C then (Y = A plus B)
   Else (Y = A plus C)
ElseIf A ≤ B then
   If B > C then (Y = B plus C)
   Else (Y = A minus B plus C)
EndIf
```

NOTA: Deberá añadirse también una salida de detección de overflow.

Tal y como vimos en teoría, para sumar números en C'2 utilizamos sumadores binarios y despreciamos el acarreo de salida. Observando en el algoritmo las operaciones que hay que realizar, podemos ver que todas pueden escribirse de la forma Y = OP1 plus OP2 plus OP3, siendo OP1 = A o 0, OP2 = B, -B (para la resta) o 0 y OP3 = C o 0, dependiendo del caso. Esto nos permite plantear una solución regular que sirve para todos los casos, basada en el uso de 2 sumadores y en la que se escogen las entradas de esos sumadores dependiendo de las condiciones del algoritmo. El circuito que tenemos que diseñar tendrá por tanto la siguiente estructura general:

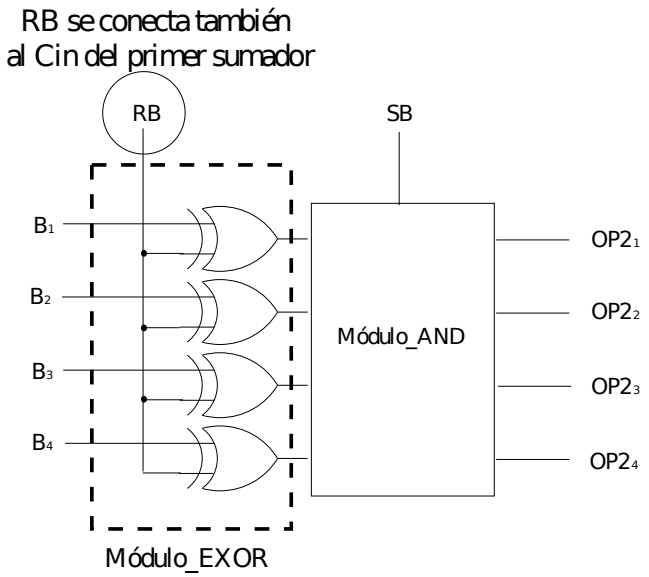


OP1 será A o 0. Para elegir entre A o 0 en función de una condición podemos utilizar puertas AND:



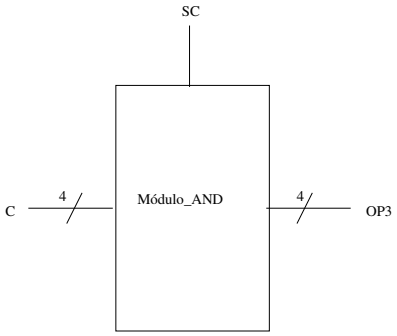
$$OP1 = \begin{cases} A & \text{si } SA = 1 \\ 0 & \text{si } SA = 0 \end{cases}$$

OP2 será B, -B o 0. Primero elegimos entre B o -B, siendo -B el C'2 de B, el cual se calcula invirtiendo B bit a bit y sumando 1 al resultado. Tal y como hemos visto en teoría en el sumador/restador, para negar una variable en función de una condición se pueden utilizar puertas EXOR. Para sumar 1 utilizaremos el Cin del sumador. A continuación elegimos entre el valor que sale de las puertas EXOR o el 0, utilizando para ello de nuevo puertas AND:



$$OP2 = \begin{cases} \text{B} & \text{si } SB = 1 \text{ y } RB = 0 \\ -B & \text{si } SB = 1 \text{ y } RB = 1 \\ 0 & \text{si } SB = 0 \end{cases}$$

OP3 será C o 0:



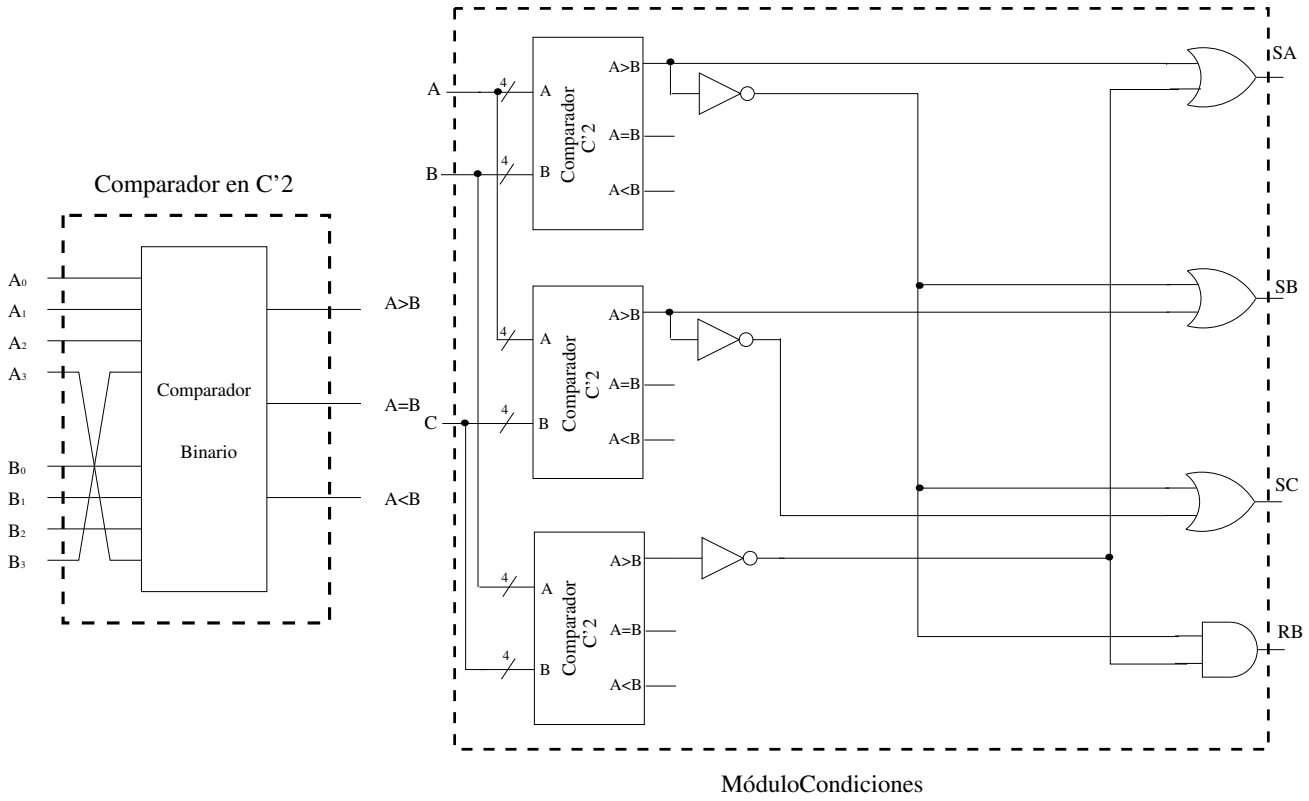
$$OP3 = \begin{cases} C & \text{si } SC = 1 \\ 0 & \text{si } SC = 0 \end{cases}$$

Para sacar las expresiones de SA, SB, SC y RB construimos una tabla de verdad:

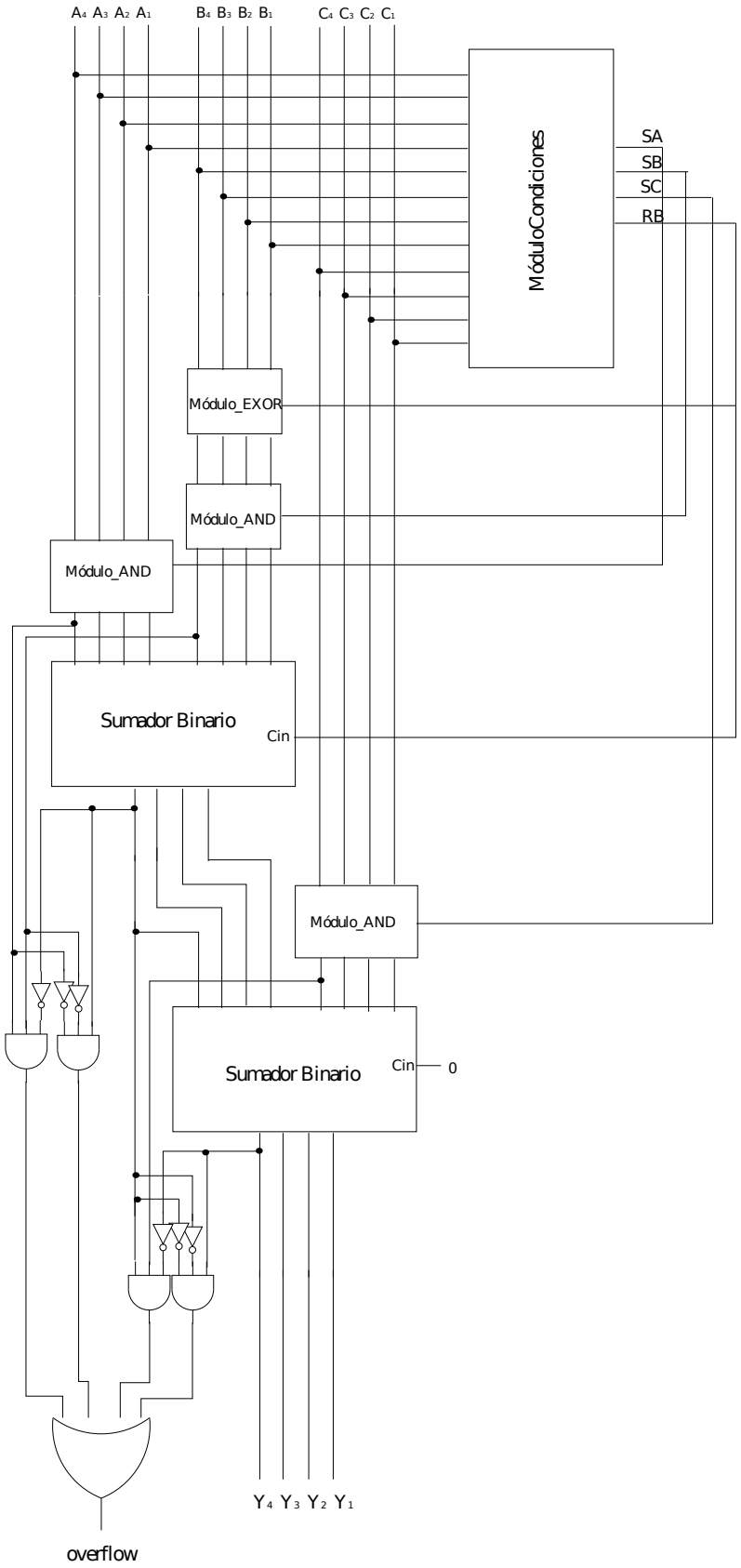
$A>B$	$A>C$	$B>C$	SA	SB	SC	RB	
0	0	0	1	1	1	1	(A-B+C)
0	0	1	0	1	1	0	(B+C)
0	1	1	0	1	1	0	
0	1	0	–	–	–	–	(imposible)
1	1	0	1	1	0	0	(A+B)
1	1	1	1	1	0	0	
1	0	1	–	–	–	–	(imposible)
1	0	0	1	0	1	0	(A+C)

$$SA = (A > B) + (\overline{B > C})$$
$$SB = (\overline{A > B}) + (A > C)$$
$$SC = (\overline{A > B}) + (\overline{A > C})$$
$$RB = (\overline{A > B}) \cdot (\overline{B > C})$$

Para implementar SA, SB, SC y RB necesitamos comparar A con B, A con C y B con C. Como hemos visto en teoría, la comparación de números en C'2 se puede realizar utilizando comparadores binarios sin más que intercambiar los bits de signo:



Finalmente, en C'2 se produce desbordamiento en la suma si los 2 sumandos son positivos y el resultado negativo, o si los 2 sumandos son negativos y el resultado es positivo. Se producirá desbordamiento si se produce desbordamiento en alguno de los sumadores:



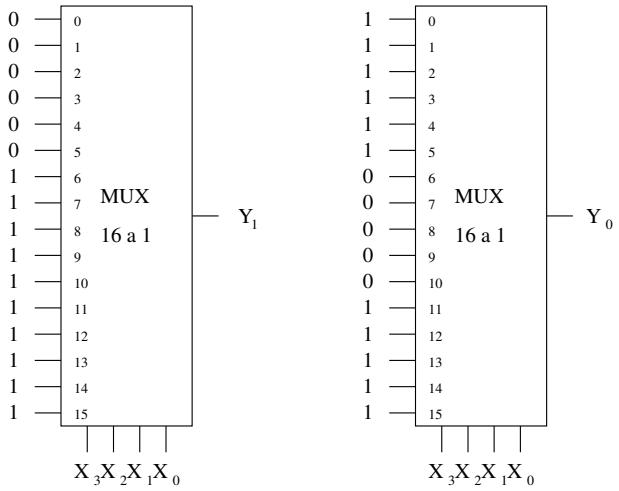
4. Diseña utilizando dispositivos MSI (decodificadores, codificadores, multiplexores o demultiplexores) y sin utilizar puertas lógicas un dispositivo que reciba como entrada números comprendidos entre el 0 y el 15 en formato binario puro y genere una salida que puede tomar 3 valores: igual a 1 si la entrada está comprendida entre el 0 y el 5, igual a 2 si la entrada está entre el 6 y el 10, e igual a 3 si la entrada está entre el 11 y el 15.

En primer lugar, como la entrada debe codificar valores entre 0 y 15 (ambos inclusive) nos llega una variable de 4 bits para representarla ($X = X_3X_2X_1X_0$). Por otra parte, la salida puede tener 3 valores diferentes (1, 2 y 3), por lo que se necesitarán 2 bits para representarla ($Y = Y_1Y_0$). Podemos crear una tabla que refleje el comportamiento pedido para el circuito:

X_3	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

Existen varios modos de implementar una función arbitraria utilizando únicamente dispositivos MSI. La más simple y directa consiste en utilizar tantos MUXes como bits tenga la función de salida (en nuestro caso 2) y con tantas entradas de control de los MUXes como bits de entrada tenga la función (en este caso 4). De esta manera, la entrada *selecciona*, en cada MUX, el bit de salida necesario (los que están en la tabla).

La solución se puede ver en la siguiente figura:



5. **Diseña un dispositivo con cuatro entradas de 6 bits en C'2,** $E0$**,** $E1$**,** $E2$ **y** $E3$**, y tres salidas:**

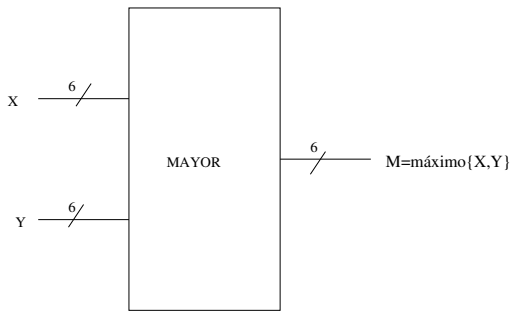
   - **Una de 6 bits en complemento a dos,** MAX**, que dé el valor máximo de las cuatro entradas:** $MAX = \max_i(Ei)$**.**
   - **Otra** P **de 2 bits en binario natural que indique la posición que ocupa el máximo, es decir, si el máximo es** $Ei \Rightarrow P = i$**.**
   - **Y una última en binario puro,** MED**, que proporcione el valor entero de la media de los valores absolutos de las cuatro entradas:** $MED = \lfloor \text{media}_i(\|Ei\|) \rfloor$**. Su número de bits ha de ser el mínimo posible para que el resultado siempre sea representable.**

APARTADO 1:

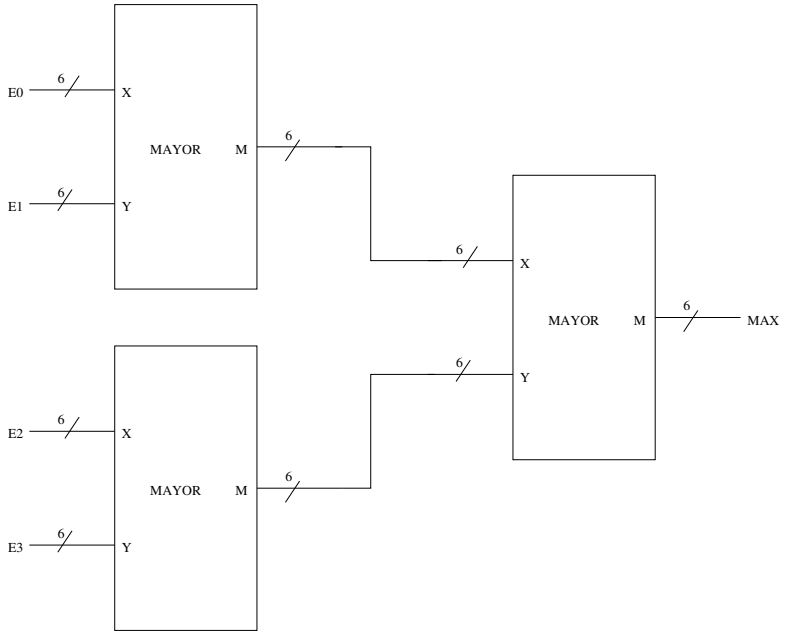
El máximo de los 4 números lo podemos calcular de la siguiente forma:

$$\text{MAX=máximo}\{E0, E1, E2, E3\} = máximo\{máximo\{E0, E1\}, máximo\{E2, E3\}\}$$

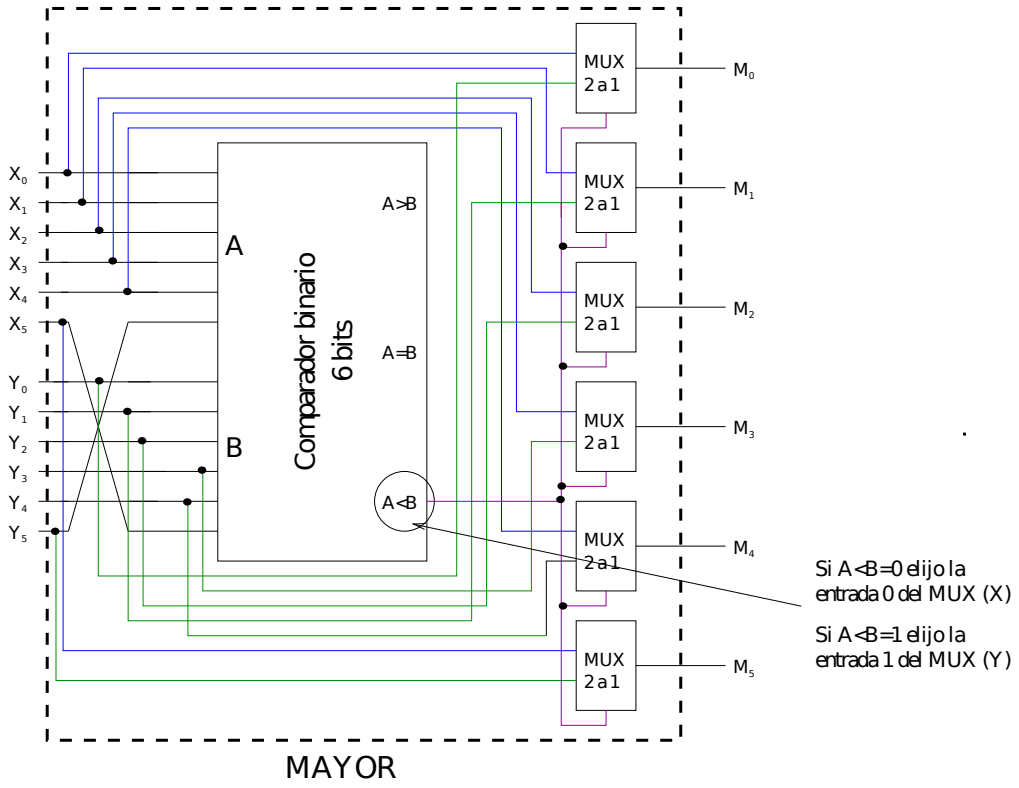
Se trata entonces de hacer un circuito digital que acepte como entrada 2 números de 6 bits en C'2 (X e Y) y proporcione como salida un número de 6 bits en C'2 (M) que se corresponda con la entrada mayor (el valor máximo):



El máximo de las 4 entradas vendrá dado entonces por:

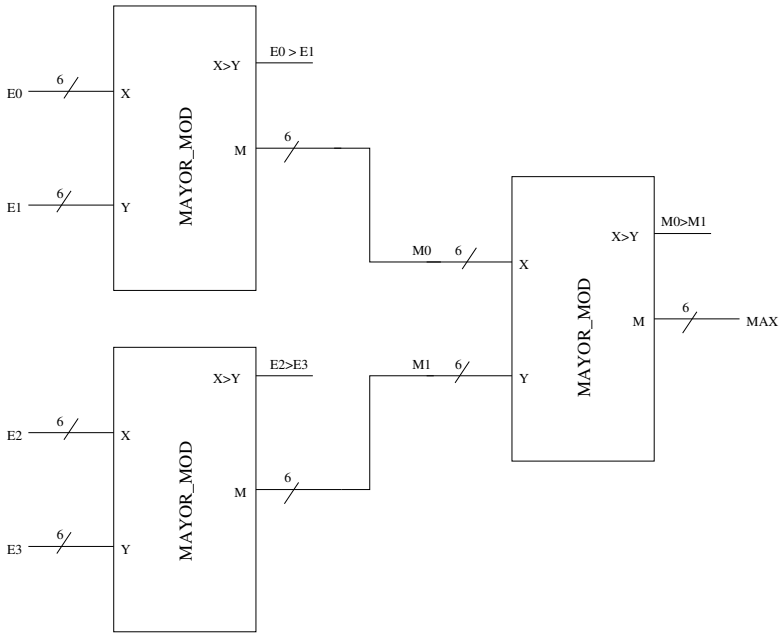


Para hacer el circuito MAYOR necesitamos comparar las 2 entradas y elegir la mayor de las dos. Como vimos en teoría, para comparar números en C'2 utilizamos comparadores binarios intercambiando los signos de las entradas. Para elegir la mayor de las dos entradas utilizamos un MUX o SELECTOR de datos. En este caso tenemos que elegir entre 2 entradas de 6 bits, necesitamos por tanto 6 MUXes (1 para cada bit) de tamaño 2 a 1 (elegimos entre 2 valores):

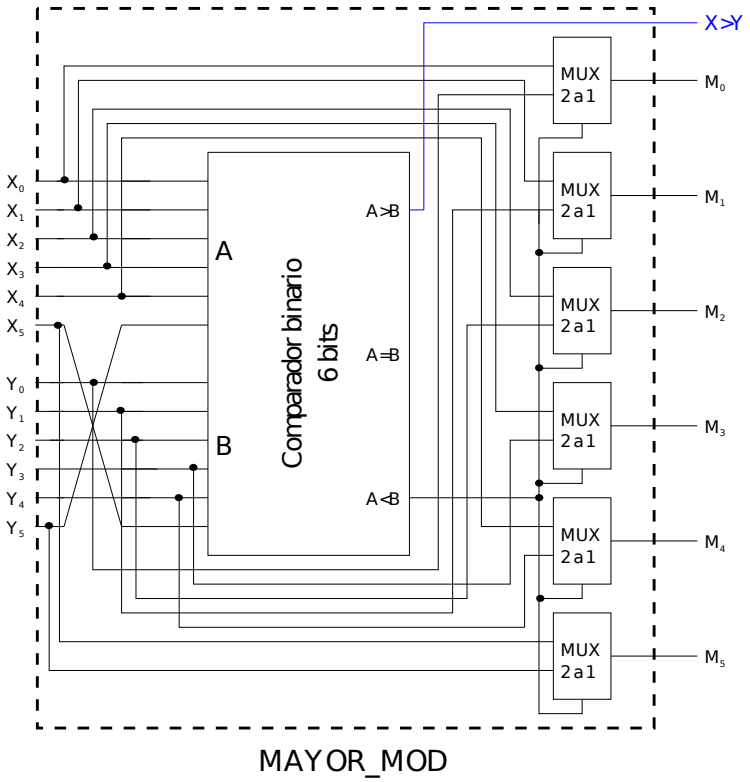


APARTADO 2:

Para poder hacer el apartado 2 tenemos que saber cuál ha sido la entrada mayor en cada una de las comparaciones. Añadimos al circuito MAYOR del apartado anterior una salida de 1 bit (X>Y) que será 1 si la entrada X es mayor que la Y y 0 en caso contrario:

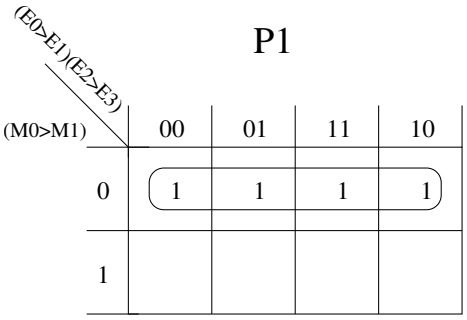


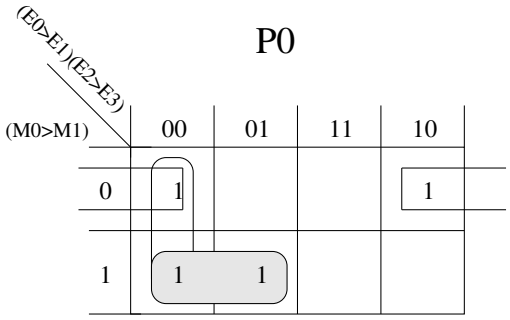
El nuevo circuito MAYOR_MOD se construye fácilmente a partir del circuito MAYOR:



Con los valores de E0>E1, E2>E3 y M0>M1 podemos determinar cuál ha sido la entrada que se corresponde con el máximo:

E0>E1	E2 >E3	M0 >M1	MAX	P1	P0
0	0	0	E3	1	1
0	0	1	E1	0	1
0	1	0	E2	1	0
0	1	1	E1	0	1
1	0	0	E3	1	1
1	0	1	E0	0	0
1	1	0	E2	1	0
1	1	1	E0	0	0

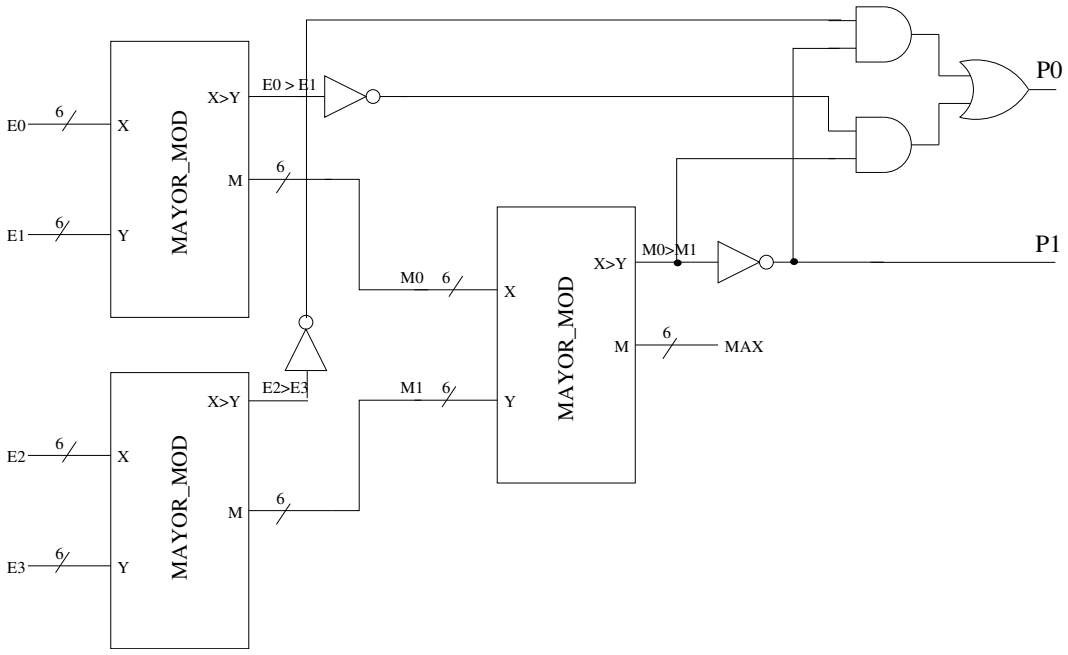




$P1=\overline{(M0>M1)}$

$P0=(M0>M1)\overline{(E0>E1)} + \begin{cases} \overline{(M0>M1)}\,\overline{(E2>E3)} \\ \overline{(E0>E1)}\,\overline{(E2>E3)} \end{cases}$

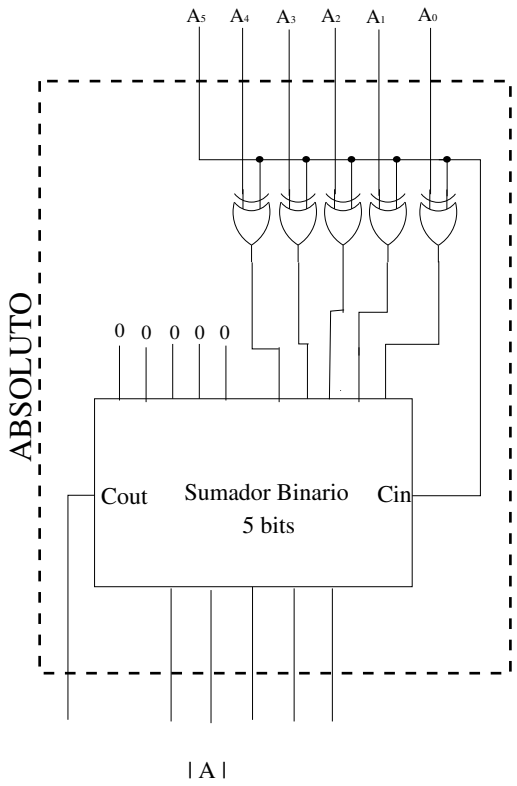
El circuito finalmente quedaría:



APARTADO 3:

$MED = \frac{|E0|+|E1|+|E2|+|E3|}{4} = \frac{(|E0|+|E1|)+(|E2|+|E3|)}{4}$

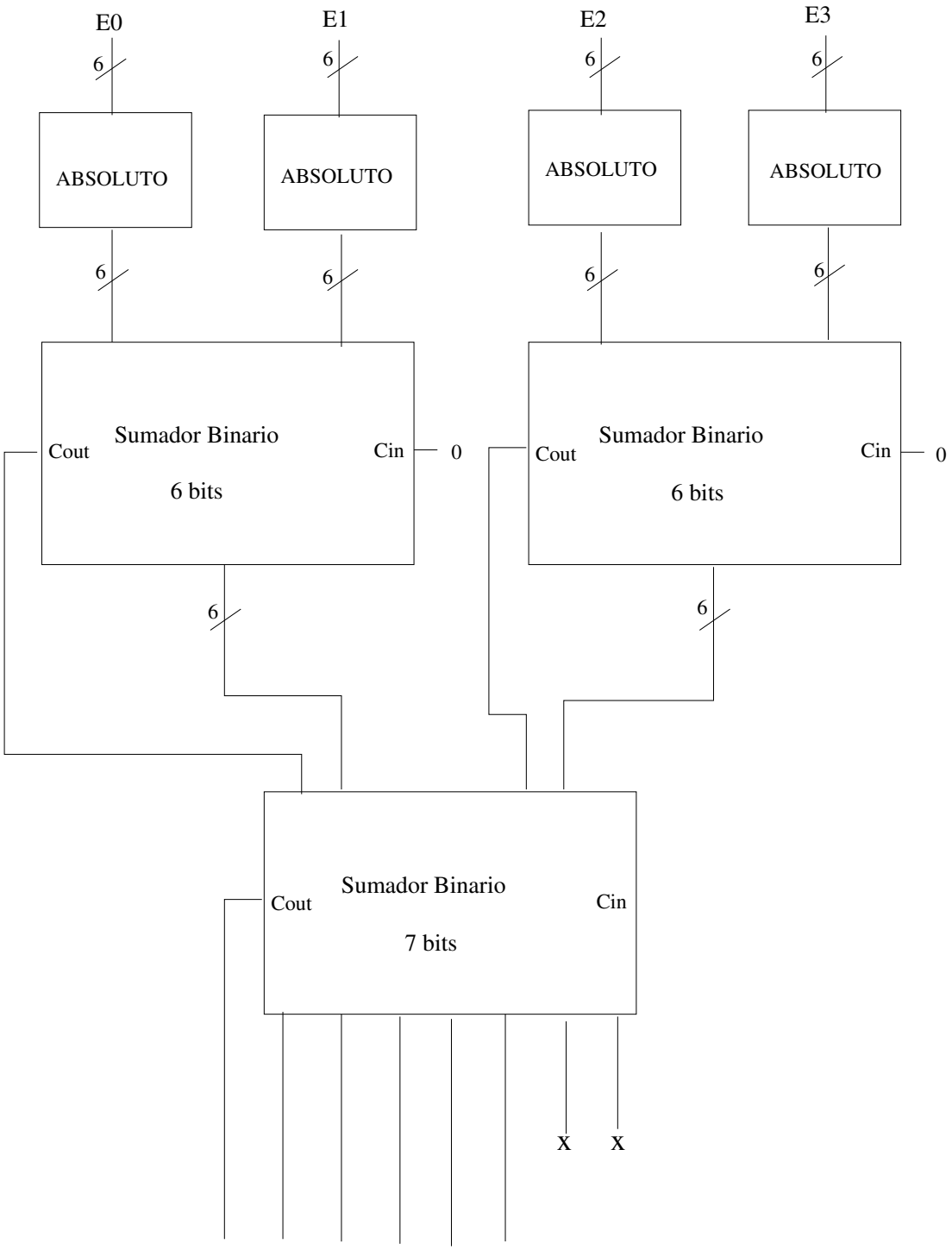
En primer lugar, vamos a construir un circuito que calcule el valor absoluto de un número A de 6 bits en C'2. El valor absoluto de un número A en C'2 es A si el número es positivo y -A (el C'2 del número) si el número es negativo. El C'2 se calcula invirtiendo el número bit a bit y sumándole 1. Tendremos, por tanto, que invertir el número A de forma condicional dependiendo de su bit de signo:



Fíjate que el resultado será un número binario puro de 6 bits, siendo el valor mayor posible el 32 (100000), correspondiente al valor absoluto del -32 (100000). Si quisiésemos representar el resultado en C'2 necesitaríamos un bit adicional para el signo, es decir, tendriamos que representar la salida con 7 bits, o bien construir una señal de detección de desbordamiento que se activaría en el caso en el que la entrada fuese -32 (el valor absoluto de -32 no es representable con 6 bits en C'2).

Una vez calculados los valores absolutos hacemos la suma. Los tenemos que ir sumando de dos en dos, necesitamos 3 sumadores binarios. Tenemos que tener en cuenta que el resultado de sumar dos números binarios puros de n bits es un número binario puro de n+1 bits (n bits de suma más 1 bit de acarreo de salida). Necesitaremos por tanto 2 sumadores binarios de 6 bits para sumar $|E0| + |E1|$ y $|E2| + |E3|$ y un sumador de 7 bits para sumar los resultados de esas 2 sumas.

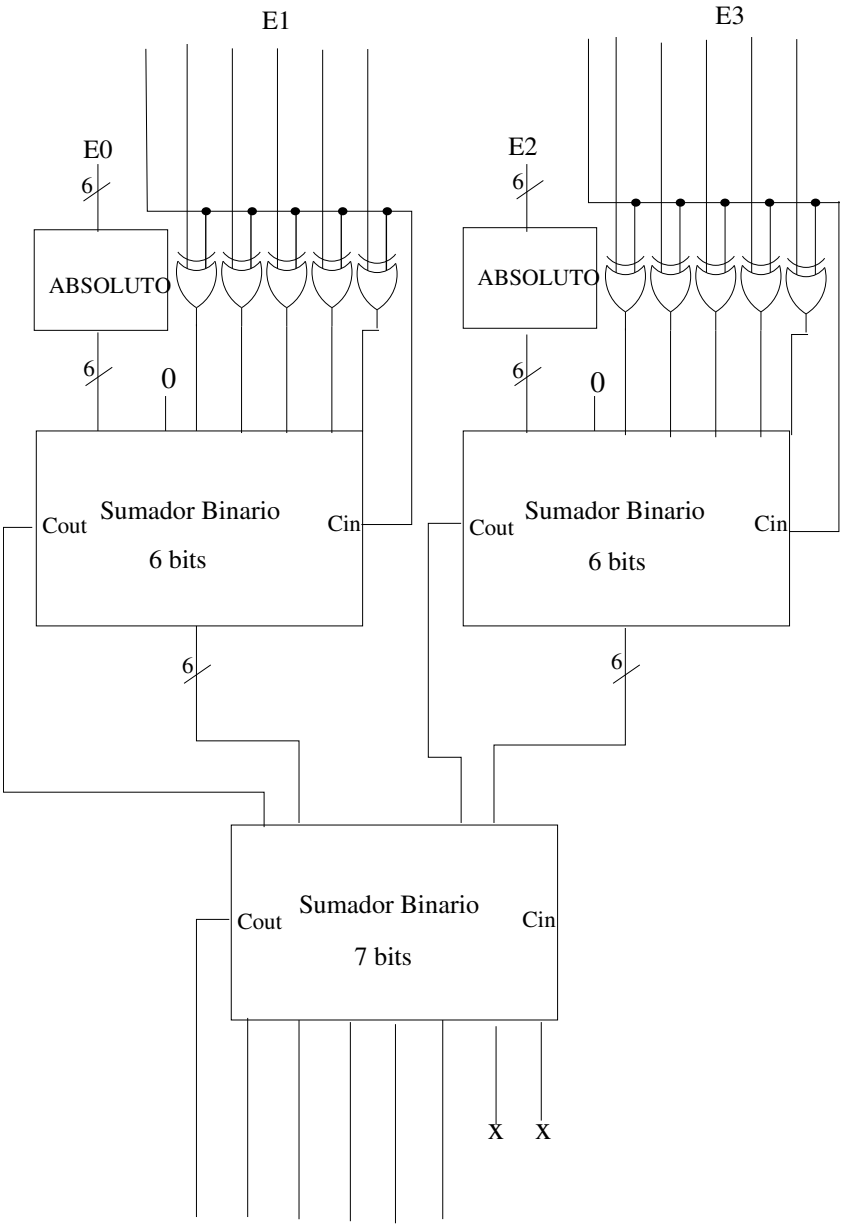
La media se calcula dividiendo el resultado de la suma entre 4, lo que equivale a desplazar el número dos posiciones a la derecha, o lo que es lo mismo, eliminar los dos bits menos significativos. El resultado es un número binario puro de 6 bits:



MED

OPTIMIZACIONES:

Podríamos reducir el número de sumadores si a la vez que calculamos el valor absoluto de E1 y E3 utilizamos el sumador para hacer las sumas parciales. Nos ahorraríamos de esta forma 2 sumadores de 5 bits:



MED