Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2018)

Tarea 2 - Pauta

Fecha de entrega: lunes 27 de agosto de 2018 a las 11:59 AM

Rúbrica: Disponible en `syllabus/Pautas/RúbricaGeneral.pdf`

Parte teórica

Pregunta 1

El **bit de paridad** es un mecanismo que se utiliza con el propósito de detectar errores durante la transmisión de información binaria. Consiste en un bit extra incluido en un mensaje binario para hacer que el número de 1s en él sea par. Una vez transmitido el mensaje, incluido el bit de paridad, se verifica en el receptor si la cantidad de bits en el mensaje es par o impar. En caso de que sea impar, el mensaje fue transmitido con un error, por lo tanto debe ser descartado.

a) Usando combinaciones de las compuertas $\wedge, \vee, \oplus,$ y $\neg$, construya los circuitos para un generador de bit de paridad para mensajes de 3 bits, y para su verificador de mensajes asociado. Este último debe indicar mediante un 1 si el mensaje es correcto y con un 0 si viene con errores. ¿Cuántos bits con errores es capaz de detectar este esquema?

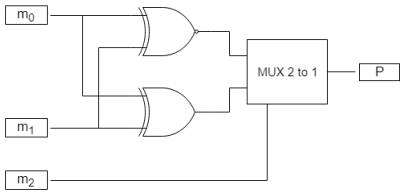
Definimos que el mensaje es $m = m_2 m_1 m_0$, que el bit de paridad es P y que este último vale 1 cuando el número de $1's$ es par, y 0 en el caso contrario. Con eso obtenemos la siguiente tabla de verdad:

m_2	m_1	m_0	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

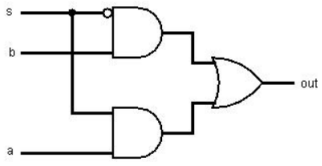
Notamos que, cuando $m_2 = 0$, estamos trabajando con un $\oplus$ y con un $\overline{\oplus}$ (`xor` negado) cuando $m_2 = 1$.

Entonces, podemos armar un circuito tal que tenga un $(m_1 \oplus m_0)$ y $(m_1 \overline{\oplus} m_0)$. Después, se puede usar un `mux` para seleccionar entre ambos y que m_2 sea el selector.

Circuito `GetParityBit`



Circuito `MUX 2 to 1`

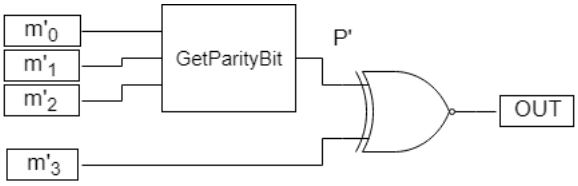


Sea el mensaje enviado $m' = m'_3 m'_2 m'_1 m'_0$, siendo m_3 el bit de paridad calculado anteriormente. Para revisar que la paridad sea correcta, queremos comparar m'_3 con el bit de paridad P' de $m'_2 m'_1 m'_0$. Si son iguales, debe retornar 1 y en el caso contrario, 0. La comparación se puede lograr con un $\overline{\oplus}$.

Tabla de verdad

m'_3	P'	`OUT`
0	0	1
0	1	0
0	0	0
0	1	1

Cicuito de comparación



Este esquema puede detectar, a lo más, 1 bit con error.

Otra alternativa: Usar sumadores de 3 bits y retornar la negación del bit menos significativo; similar a lo que será mostrado en la pregunta (c).

b) Usando sólo combinaciones de uno de los siguientes tipos de compuerta, $\wedge, \vee, \oplus$ y $\neg$, construya los mismos circuitos del ítem anterior. Es decir, si por ejemplo escoge hacerlo con $\wedge$, sólo puede emplear ese tipo de compuertas en su diseño. No es necesario que ambos circuitos utilicen el mismo tipo de compuerta.

Respuesta: Basta que la tabla de verdad del circuito implementado cumpla las mismas propiedades de las dadas en la pregunta anterior.

A continuación, se deja una alternativa:

Circuito `GetParityBit`

$$(m_2 \oplus m_1) \oplus m_0$$

Circuito de comparación

$$(m'_3 \oplus m'_2) \oplus (m'_1 \oplus m'_0)$$

c) Generalice el diseño del ítem anterior, para mensajes de N bits, con $N >= 3$.

Respuesta: Tomando B_p como el bit de paridad y B_v como el bit verificador, tenemos la siguiente alternativa según lo expuesto en (b):

$$B_p = m_N - 1 \oplus m_N - 2 \oplus ... \oplus m_1 \oplus m_0$$

$$B_v = m'_N \oplus m'_N - 1 \oplus ... \oplus m'_1 \oplus m'_0$$

Realizando la operación de izquierda a derecha, o de derecha a izquierda, obtenemos el resultado esperado.

Note que esto es equivalente a sumar los N bits del mensaje mediante *half-adders* en cadena (ignorando el *carry*), ya que estos se componen de compuertas $\oplus$.

Pregunta 2

Un comparador de números es un circuito que, dados dos números A y B en representación posicional, indica cuál es el mayor, o si estos son iguales. El circuito posee tres salidas, donde la primera entrega un 1 sólo si A es el mayor, la segunda un 1 sólo si ambos son iguales, y la tercera un 1 sólo B es mayor.

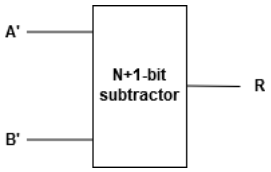
a) Diseñe un comparador de números naturales de N bits.

Respuesta: Es de gran utilidad implementar un restador que tenga de entradas los números A y B. Ahora, como no contamos con representaciones negativas, es de utilidad que nuestro restador sea de $N+1$ bits y, para A, B definimos A', B' tales que tienen un bit adicional igual a 0 en su posición más significativa. Ahora, Considerando $R = A' - B'$, podemos realizar el siguiente análisis del resultado a partir del bit más significativo:

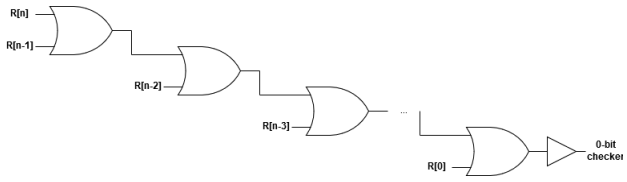
- Si $R_N = 1$, el resultado es negativo (lo que implica que A era menor a B).
- Si $R_N = 0$, el resultado es positivo o cero. Para determinar si es 0, basta con utilizar una secuencia de compuertas $\vee$ para R, de forma que retorna 0 si, y solo si todos los bits en R son iguales a cero.

Entonces, nuestro circuito se ve como sigue:

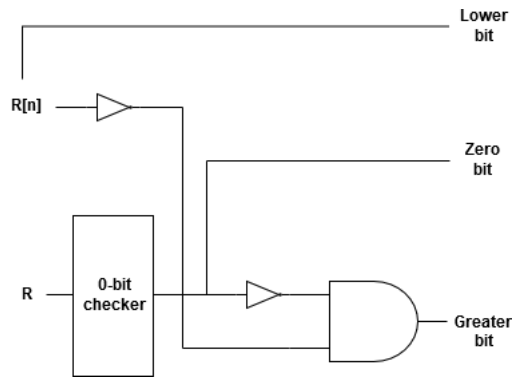
Restador de $N+1$ bits



Circuito para detectar $R = 0$



Circuito final



De esta forma, tenemos que:

- `0-bit checker` es un bit igual a 1 si, y solo si todos los bits de R son iguales a cero (*i.e.* $A = B$).
- `Lower bit` es igual a 1 si, y solo si el bit más significativo del resultado es igual a 1 (lo que implica $A < B$).
- `Greater bit` es igual a 1 si, y solo si el bit más significativo del resultado es igual a 0 (lo que implica que $A \geq B$) **y, además,** existe al menos un bit igual a 1 en el resultado (*i.e.* $A \neq B \rightarrow A > B$).
- `Zero bit` es igual a 1 si, y solo si el bit más significativo del resultado es igual a 0 (lo que implica que $A \geq B$) **y, además,** todos los bits del resultado son iguales a cero (*i.e.* $A = B$).

De esta forma, nuestro circuito final cumple con lo pedido.

b) Diseñe un comparador de números enteros de N bits.

Ahora, nuestros números pueden ser positivos o negativos, por lo que no es necesario añadir un bit adicional para operar. Aquí, entonces, tenemos cuatro casos:

1. $A \geq 0$, $B \geq 0$.
2. $A \geq 0$, $B < 0$.
3. $A < 0, B \geq 0$.
4. $A < 0, B < 0$.

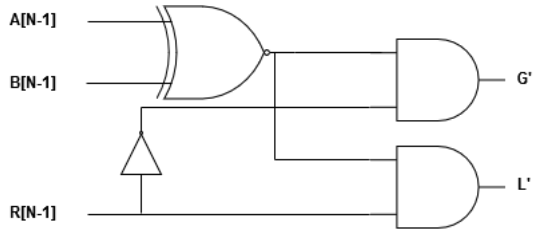
En los casos 2 y 3 podemos ver que basta con identificar que sus bits de signo sean distintos, haciendo `larger bit = 1` o `lower bit = 1`, según corresponda. Por otra parte, vemos que para los casos 1 y 4, independiente del signo que compartan A y B, se cumple que:

$$A - B > 0 \rightarrow A > B \rightarrow (A - B)_{N-1} = 0$$

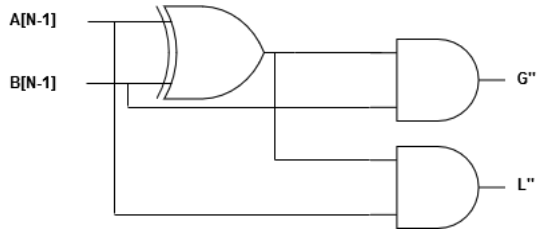
$$A - B < 0 \rightarrow A < B \rightarrow (A - B)_{N-1} = 1$$

Como el caso en el que $A = B$ sigue siendo el mismo, el siguiente conjunto de circuitos cumple lo pedido:

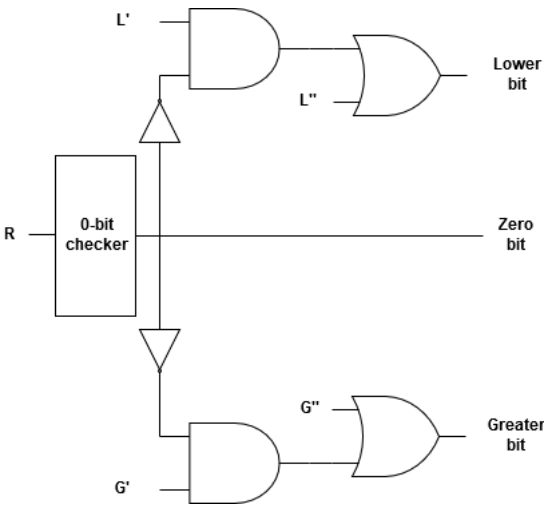
Circuito para los casos 1 y 4



Circuito para los casos 2 y 3



Circuito final

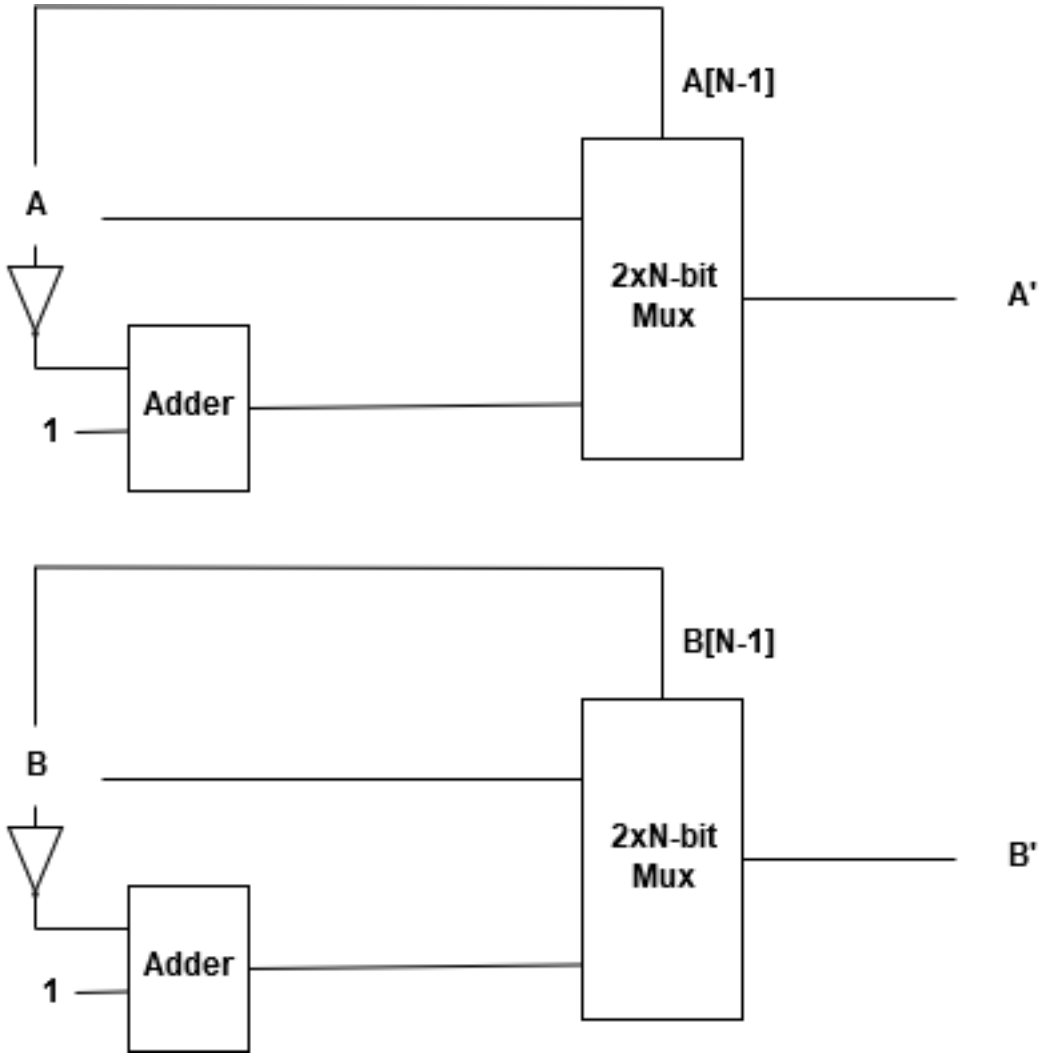


Para los casos 2 y 3, simplemente hacemos un circuito tal que `Greater bit = 1` y `Lower bit = 0` si A es positivo y B negativo, mientras que `Greater bit = 0` y `lower bit = 1` si A es negativo y B positivo (comprobamos que los signos de A y B sean distintos con la compuerta `XOR` para que este caso sea válido). Por otra parte, para los casos 1 y 4 necesitamos hacer la resta para poder revisar el signo resultante, el que verificamos que sea positivo o negativo para determinar si $A > B$ o $A < B$ (comprobamos que los signos de A y B sean iguales con la compuerta `XNOR` para que este caso sea válido). Notar que en este caso hacemos uso de un restador de N bits (y no de $N+1$ como en el caso anterior). Finalmente, usamos el mismo `0-bit checker` para determinar si $A = B$, lo que implica `Zero bit = 1`.

c) Diseñe un comparador del valor absoluto de números enteros de N bits.

Básicamente, basta con usar el circuito diseñado en (a) (al ser ambos positivos) ó en (b), pero seleccionando, para cada entrada, el número original o su inverso aditivo (correspondiente al complemento a 2 del valor), dependiendo de su signo. Para ello, basta con utilizar un `mux` de dos entradas de N bits, cuyo bit de selección sea el bit más significativo del número (*i.e.* su bit de signo).

Circuito de valor absoluto



En este caso, A' y B' son los valores absolutos de A y B, respectivamente.

Finalmente, ingresando esto a cualquiera de los dos circuitos, logramos el mismo resultado. Notar que para (a), no obstante, habría que hacer uso de un restador de N y no de $N+1$ bits.

Parte programación

Tests para la parte programación:

1. **Caso base 1:** Compuerta $\neg$ (`not.json`).
2. **Caso base 2:** Compuerta $\wedge$ (`and.json`).
3. **Caso base 3:** Compuerta $\vee$ (`or.json`).
4. **Caso base 4:** Compuerta $\oplus$ (`not.json`).
5. **Caso medio 1:** Circuito en serie (`testSeq.json`).
6. **Caso medio 2:** Circuito en paralelo (`testPar.json`).
7. **Caso complejo 1:** Mezcla de todas las compuertas (`testComp1.json`).
8. **Caso complejo 2:** Mezcla de todas las compuertas más grande (`testComp2.json`).

Se simuló, para cada combinación de *inputs*, el *output* resultante del circuito original y del retornado por su programa. La asignación de puntaje se realizó según lo estipulado en la rúbrica general.

Parte práctica

Tests para la parte práctica:

1. **Caso cero:** ambos registros son cero, la respuesta es 0
2. **Revisar correspondencia de partes 1:** `[0001 0001 0001 0001] x [0001 0000 0000 0000] =>` 1
3. **Revisar correspondencia de partes 2:** `[0001 0001 0001 0001] x [0000 0001 0000 0000] =>` 1
4. **Revisar correspondencia de partes 3:** `[0001 0001 0001 0001] x [0000 0000 0001 0000] =>` 1
5. **Revisar correspondencia de partes 4:** `[0001 0001 0001 0001] x [0000 0000 0000 0001] =>` 1
6. **Caso máximo:** `[1111 1111 1111 1111] x [1111 1111 1111 1111] =>` $(384)_{16}$
7. **Caso normal 1:** `[0011 0100 0101 0110] x [0001 0010 0011 0100] =>` $(32)_{16}$
8. **Caso normal 2:** `[0011 0100 0101 0110] x [0000 0001 0000 0001] =>` $(A)_{16}$