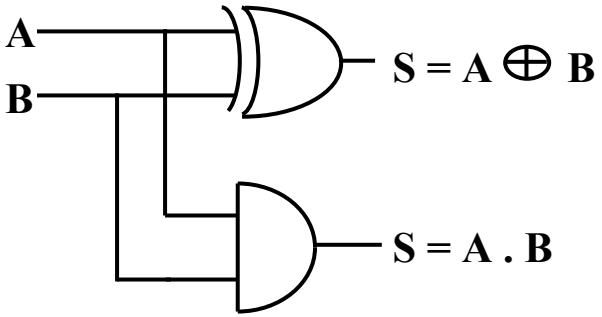
SUMA BINARIA

SEMI-SUMADOR

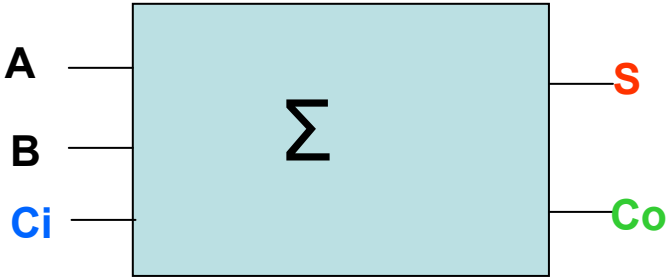
SUMANDOS		SUMA	ACARREO
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



SUMADOR – TOTAL

Ejemplo de suma

Ci	1	1	0	0	1
Ai	1	1	1	0	1
Bi		1	0	0	1
Si	1	0	0	1	1	0
Co	1	1	0	0	1	

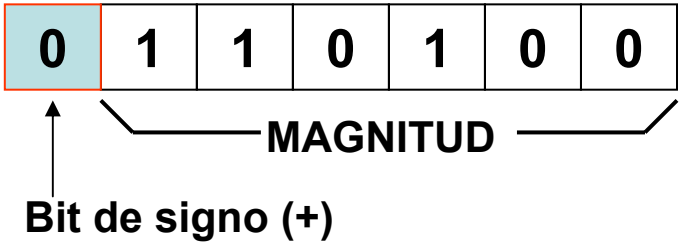


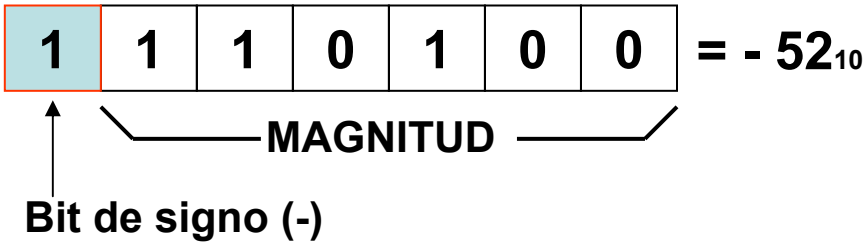
Ci	A	B	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = \Sigma 1,2,4,7 = Ci \oplus A \oplus B$

$Co = \Sigma 3,5,6,7 = AB + CiA + CiB$

NUMEROS CON SIGNO





COMPLEMENTO A 1

Se deben cambiar los 0 por 1 y los 1 por 0, Ej. 1 0 1 1, numero real

0 1 0 0, Complemento a 1

COMPLEMENTO A 2

Es el complemento a 1 y se le suma 1,

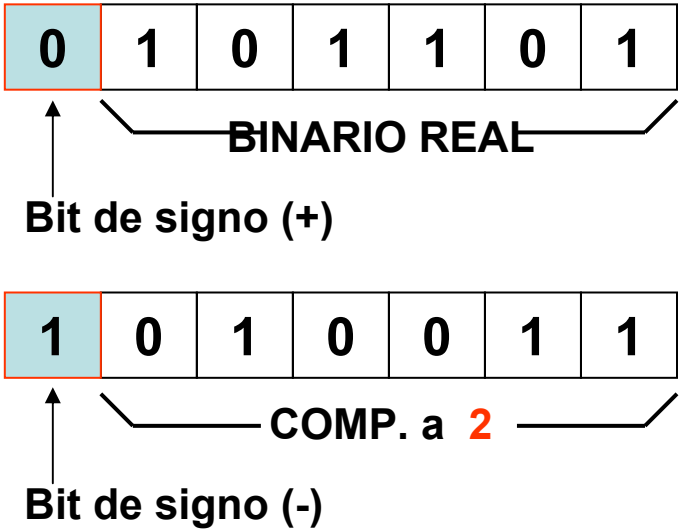
Ej, 1 0 1 1, numero real

0 1 0 0, Complemento a 1

\+ 1

0 1 0 1, Complemento a 2

REPRESENTACION DE NUMEROS CON SIGNO



EJEMPLOS: (UTILIZAR CINCO BITS INCLUYENDO EL SIGNO)

+ 13 → 0 1 1 0 1

- 9 → 1 0 1 1 1

+ 3 → 0 0 0 1 1

- 3 → 1 1 1 0 1

DESBORDAMIENTO (OVERFLOW) ARITMETICO

EJEMPLO DE SUMA:

+ 3	0 0 0 1 1
+ 5	0 0 1 0 1
3 + 5	0 1 0 0 0

EN LOS EJEMPLOS ANTERIORES VIMOS QUE LA SUMA BINARIA DE 4 BITS MAS UNO DE SIGNO (CINCO EN TOTAL) NO HABIA ACARREO HACIA LA QUINTA POSICIÓN. VEAMOS QUE OCURRE SI REALIZAMOS UNA OPERACION QUE EXCEDA

+ 9	0	1 0 0 1
+ 8	0	1 0 0 0
9 + 8	1	0 0 0 1

SIGNO INCORRECTO

MAGNITUD INCORRECTA

LA RESPUESTA CORRECTA ES + 17 PERO TAL MAGNITUD REQUIERE MAS DE 4 BITS Y POR LO TANTO PROVOCA UN DESBORDE (OVERFLOW)

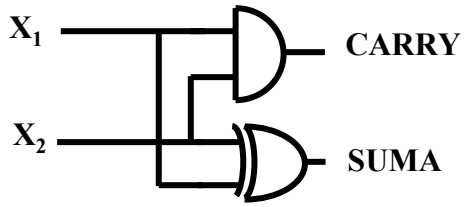
MODELOS VHDL

Desarrollemos el código VHDL para un semisumador de un bit

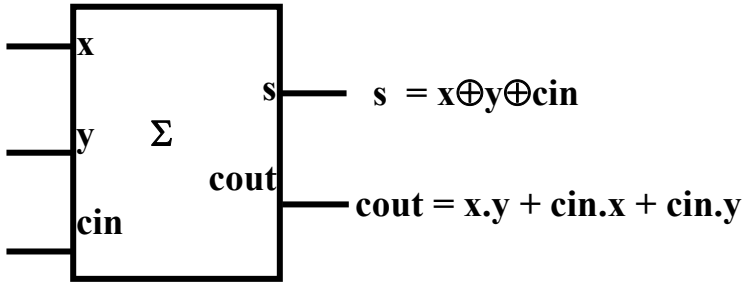
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY semisumador IS
                PORT (x1, x2,  :      IN  STD_LOGIC  ;
                                Suma,carry:  OUT  STD_LOGIC);
END semisumador;

      ARCHITECTURE semi  OF semisumador   IS
      BEGIN
                suma<= x1 XOR x2;
                carry<= x1 AND x2;
      END semi;
```



SUMADOR TOTAL




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY fulladd IS
        PORT (cin, x, y :  IN     STD_LOGIC;
                  s, cout        :  OUT  STD_LOGIC);
END fulladd;
    ARCHITECTURE LogicFunc  OF fulladd   IS
    BEGIN
        s <= (x XOR y XOR cin);
        cout<= (x AND y) OR (cin AND x) OR ( cin AND Y);
END LogicFunc;
```

Representación de Números en VHDL

Así como en circuitos lógicos un número es representado por señales en un conexionado de múltiples cables, un número en VHDL es representado como una señal de datos multibit. Un ejemplo de esto es:

El tipo de dato STD_LOGIC_VECTOR representa un arreglo lineal del dato STD_LOGIC.

La declaración SIGNAL define a C como una señal STD_LOGIC de 3 bits. Si por ejemplo asignamos

C <= “100”;

Ello significa que C(1) = 1 , C(2) = 0 y C(3) = 0

Otro tipo de declaración SIGNAL es:

SIGNAL X: STD_LOGIC_VECTOR (3 DOWNTO 0);

SENTENCIAS DE ASIGNACION ARITMETICA

Lo que define a X como una señal de STD_LOGIC_VECTOR de 4 bits, especificando que el bit mas significativo de X es designado X(3) y el menos significativo X(0)

Ejemplo: X<= “1100”

significa: X(3) = 1, X(2)=1, X(1) = 0 , X(0)= 0

Sentencias de Asignación Aritmética

Si definimos:

```
SIGNAL X, Y, S: STD_LOGIC_VECTOR(15 DOWNTO 0);
```

Luego las sentencia de asignación aritmética

```
S<= X + Y;
```

Representa a un sumador de 16 bits

MODELO VHDL – SUMADOR 16 BITS

Agui vemos el uso de estas sentencias y se ha incluido el paquete *std_logic_signed* para permitir el uso del operador adición (+)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
        PORT ( X, Y   : IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
                 S    : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0) );
END adder16;


ARCHITECTURE Behavior OF adder16 IS
BEGIN
        S <= X + Y ;
END Behavior;
```

OBSERVE QUE NO SE INCLUYEN LAS SEÑALES DE ACARREO DE ENTRADA NI DE SALIDA NI OVERFLOW -

CONSIDERACIONES – SUMADOR DE 16 BITS

El código anterior no incluye las señales Cin y Cout y tampoco el overflow. Ello se subsana con el código que se propone a continuacion

El bit 17 designado *sum* , es usado por el Cout desde la posición 15 del sumador.
El término entre paréntesis (0&X), es que el 0 esta concatenado al bit 16 de la señal X para crear el bit 17.

En VHDL el operador & se llama operador *concatenate*. Este operador no tiene el significado de la funcion AND. La razón de introducirlo en este código es que VHDL requiere que al menos uno de los operandos X o Y tengan el mismo numero de bits que el resultado

Otro detalle es la sentencia:

```
S <= Sum (15 DOWNTO 0);
```

Esta sentencia asigna los 16 bits menos significativos de *Sum* a la salida S

MODELO 2 –VHDL – SUMADOR DE 16 BITS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
        PORT ( Cin              : IN     STD_LOGIC;
               X, Y             : IN     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               S                : OUT  STD_LOGIC_VECTOR(15 DOWNTO 0) ;
             Cout, Overflow     : OUT  STD_LOGIC) ;
END adder16;

ARCHITECTURE Behavior OF adder16 IS
        SIGNAL Sum : STD_LOGIC_VECTOR(16 downto 0) ;
BEGIN
        Sum <= ('0' & X) + Y + Cin ;
        S <= Sum(15 Downto 0) ;
        Cout <= Sum(16) ;
        Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

Sumador con señales de carry y overflow

CONSIDERACIONES SUMADOR CON CARRY Y OVERFLOW

La sentencia : Cout <= Sum (16); asigna el carry-out de la adición ,(*sum (16)*), a la señal carry-out, (*Cout*)

La expresión de overflow es Cn-1 XOR Cn , en nuestro caso Cn = *Sum (16)* y por otra parte

$$Cn\text{-}1 = X(15) \oplus Y(15) \oplus S(15)$$

El uso del paquete *std_logic_signed* permite que las señales STD_LOGIC sean utilizadas con operadores aritméticos. Este paquete realmente usa otro paquete , el *std_logic-arith*, el que define dos tipos de datos llamados SIGNED Y UNSIGNED, para uso en circuitos aritméticos que tratan con numeros con y sin signo.

MODELO VHDL – SUMADOR – PAQUETE ARITMETICO

El codigo anterior puede ser reescrito para usar directamente el paquete *std_logic-arith*, tal como se observa en la Fig. 24

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;

ENTITY adder16 IS
        PORT ( Cin                      : IN    STD_LOGIC;
               X, Y                     : IN    SIGNED(15 DOWNTO 0) ;
               S                        : OUT   SIGNED(15 DOWNTO 0) ;
               Cout, Overflow  : OUT  STD_LOGIC) ;
END adder16;


ARCHITECTURE Behavior OF adder16 IS
        SIGNAL Sum : SIGNED(16 downto 0) ;
BEGIN
        Sum <= ('0' & X) + Y + Cin ;
        S <= Sum(15 DOWNTO 0) ;
        Cout <= Sum(16) ;
        Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

Uso del paquete aritmetico

RESTA BINARIA

LA RESTA DE DOS NUMEROS BINARIOS ES EQUIVALENTE A UNA SUMA ALGEBRAICA.

SEA:

$B = 1\ 0\ 1\ 1$

$\overline{B} = 0\ 1\ 0\ 0$

$B + \overline{B} = 1\ 1\ 1\ 1$

$+\quad 1$

$B + \overline{B} + 1 = 1\ 0\ 0\ 0\ 0 = 2^n$

(Bit de overflow)

$$B = 2^n - \overline{B} - 1$$

$$\overline{B} = 2^n - B - 1$$

$(A - B) > 0$:

$A - B = A - (2^n - \overline{B} - 1)$

$A - B = A + \overline{B} + 1 - 2^n$

Bit de over flow

$A = 1\ 1\ 0\ 0 \rightarrow A = 1\ 1\ 0\ 0$

$B = 0\ 1\ 1\ 1 \rightarrow \overline{B} = 1\ 0\ 0\ 0$

$\overline{A} + B = 101\ 0\ 0$

$+\quad 1$

Overflow

$\overline{A} + B + 1 = 1\ 0\ 1$

LA EXISTENCIA DEL OVERFLOW INDICA QUE EL RESULTADO ES POSITIVO O SEA A>B. SI NO EXISTIERA OVERFLOW EL RESULTADO SERIA NEGATIVO (A<B)

RESTA < 0

(A – B) < 0:

$$A - B = A + \overline{B} + 1 - 2^n$$

$$A + \overline{B} = 2^n - 1 + (A - B)$$

YA QUE (A – B) = - D

$$A + \overline{B} = 2^n - 1 - D = \overline{D}$$

$$D = \overline{A + \overline{B}}$$

Ejemplo:

$$A = 0\,1\,1\,1 \longrightarrow A = 0\,1\,1\,1$$

$$B = 1\,1\,0\,0 \longrightarrow \overline{B} = \underline{0\;\;0\,1\,1}$$

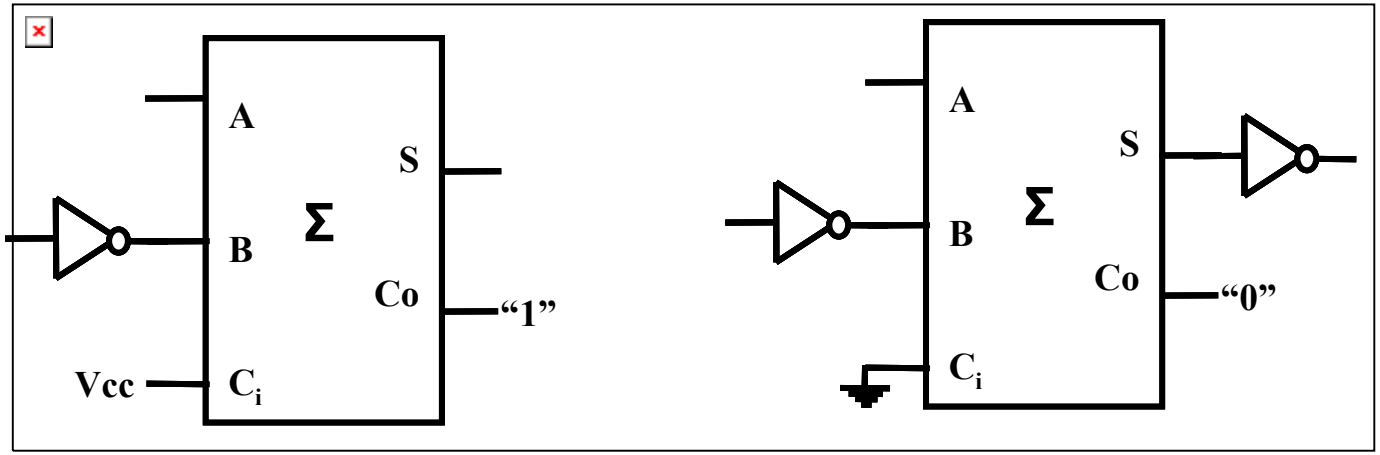
$$A + \overline{B} = 1\,0\,1\,0 \longrightarrow \text{(NO HAY OVERFLOW)}$$

$$\overline{\overline{A + \overline{B}}} = 0\,1\,0\,1 = (A - B)$$

CIRCUITOS REPRESENTATIVOS

$(A-B) > 0$

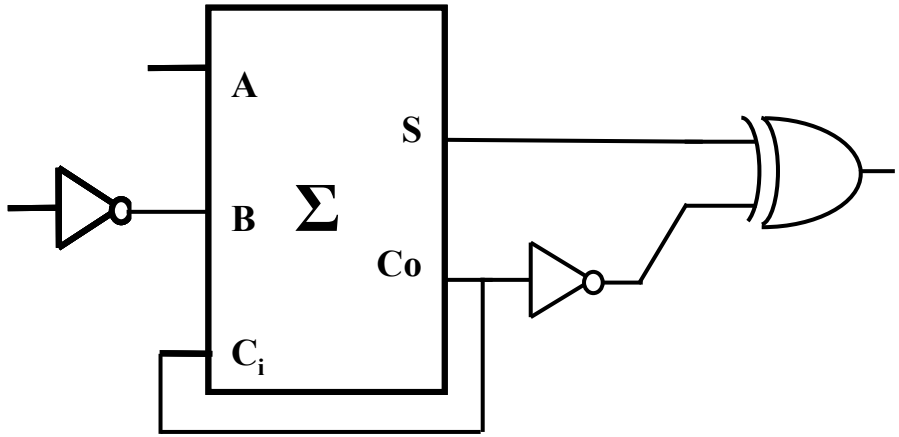
$(A-B) < 0$



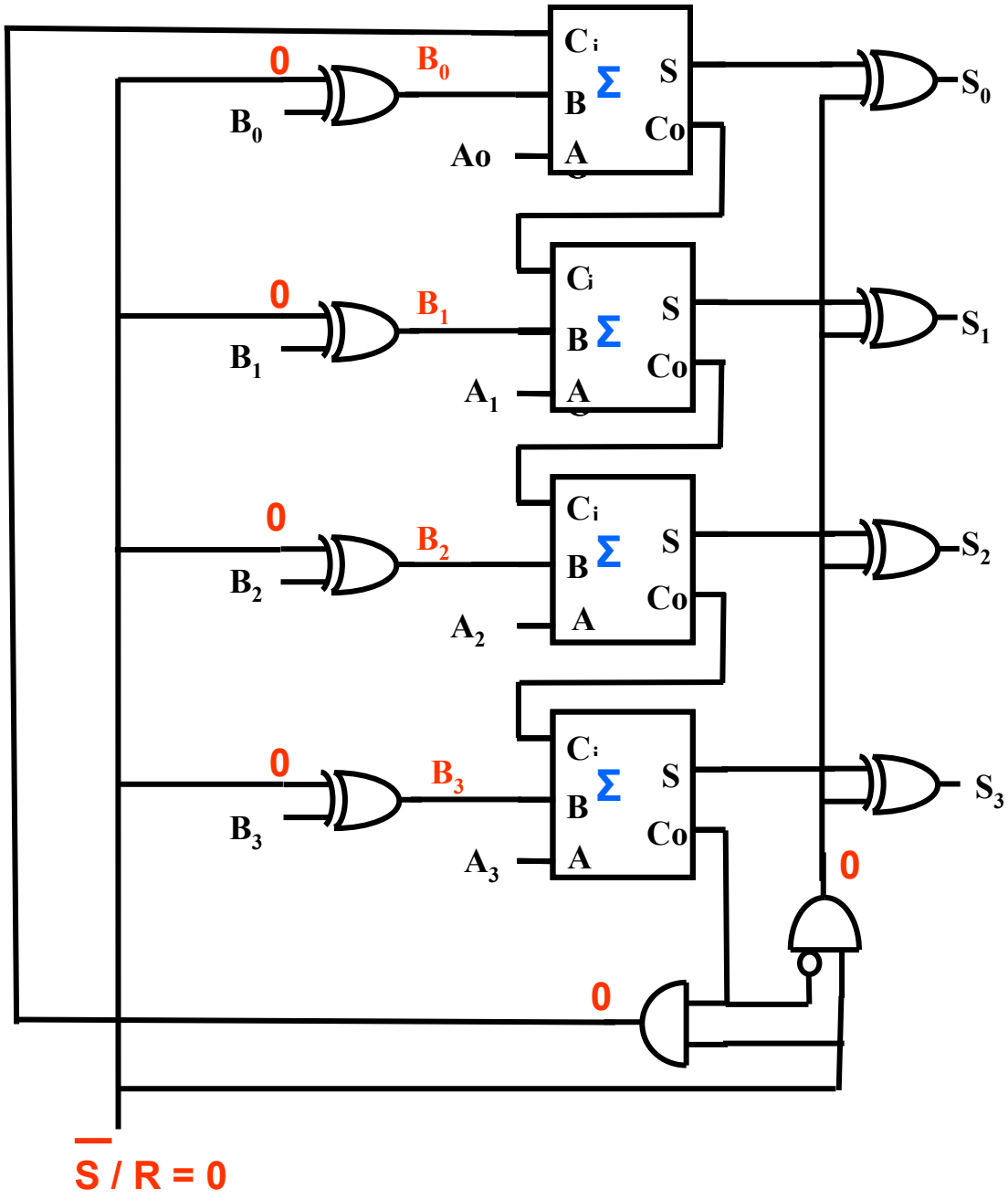
RESTADOR PARA $(A - B) > 0$ y $(A - B) < 0$

$A \oplus 1 = \overline{A}$

$A \oplus 0 = A$



SUMADOR/RESTADOR DE 4 BITS



A + B

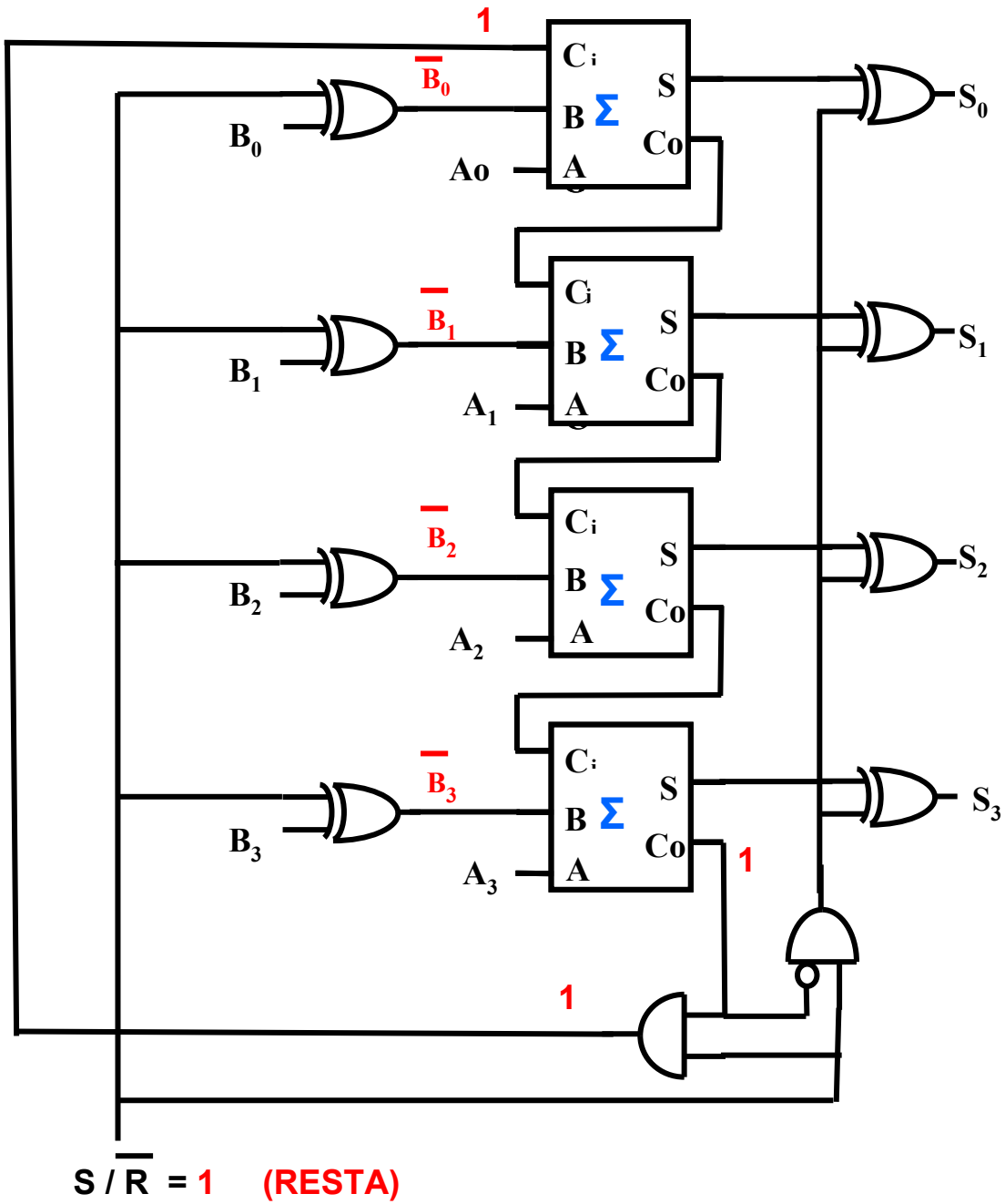
$\overline{S}$ / R = 0 → SUMA

$\overline{S}$ / R = 1 → RESTA

$B \oplus 1 = \overline{B}$

$B \oplus 0 = B$

RESTADOR DE 4 BITS



(A - B) > 0

$B \oplus 1 = \overline{B}$

$B \oplus 0 = B$

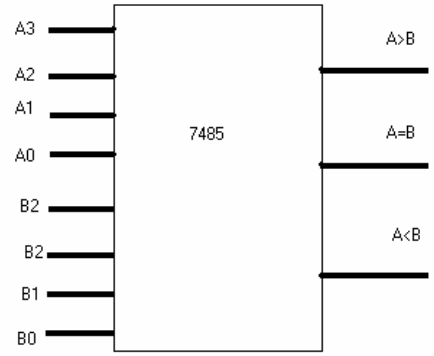
$\overline{S}$ / R = 0 → SUMA

$\overline{S}$ / R = 1 → RESTA

COMPARADOR

- COMPARADOR DE MAGNITUD

A	B	$A \neq B$	$A = B$	$A > B$	$A < B$
		$A \oplus B$	$\overline{A \oplus B}$	$A \cdot \overline{B}$	$\overline{A} \cdot B$
0	0	0	1	0	0
0	1	1	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0



MODELO VHDL - COMPARADOR

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY compare IS
        PORT ( A, B            : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            AeqB, AgtB, AltB   : OUT STD_LOGIC ) ;
END compare ;

ARCHITECTURE Behavior OF compare IS
BEGIN
    AeqB <=  '1'  WHEN A = B   ELSE  '0';
    AgtB <=  '1'  WHEN A > B   ELSE '0';
    AltB  <=  '1'  WHEN A < B   ELSE '0';
END Behavior ;
```

Codigo VHDL para un comparador de 4 bits

MODELO2 VHDL COMPARADOR

Otra forma de especificar el circuito es incluir la librería denominada *std_logic_arith* . En ambos casos las señales A y B deberían estar definidas con el tipo UNSIGNED, mas bien que STD_LOGIC_VECTOR. Si nosotros queremos que el circuito trabaje con numeros con signo, las señales A y B deberían ser definidas con el tipo SIGNED.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;

ENTITY compare IS
        PORT ( A, B                    : IN     SIGNED(3 DOWNTO 0) ;
                    AeqB, AgtB, AltB   : OUT STD_LOGIC ) ;
END compare ;
 ARCHITECTURE Behavior OF compare IS
BEGIN
      AeqB <=   '1'  WHEN A = B ELSE `0`;
      AgtB <=   '1'  WHEN A > B ELSE `0`;
      AltB  <=  '1'  WHEN A < B ELSE `0`;
END Behavior ;
```

CODIFICACION SECUENCIAL
SENTENCIA PROCESS

ES UNA SENTENCIA CONCURRENTE (SE EJECUTA EN PARALELO) QUE ENGLOBA UN CONJUNTO DE SENTENCIAS QUE SE EJECUTAN SECUENCIALMENTE.

TANTO EL SIMULADOR COMO EL SINTETIZADOR INTERPRETAN AL BLOQUE PROCESS COMO SI SE TRATASE DE UNA SOLA SENTENCIA.

TODO PROCESO CONLLEVA UNA LISTA DE SENSIBILIDAD (opcional), QUE ES UN CONJUNTO DE SEÑALES CUYO CAMBIO ACTIVA LA EJECUCIÓN DEL PROCESO.

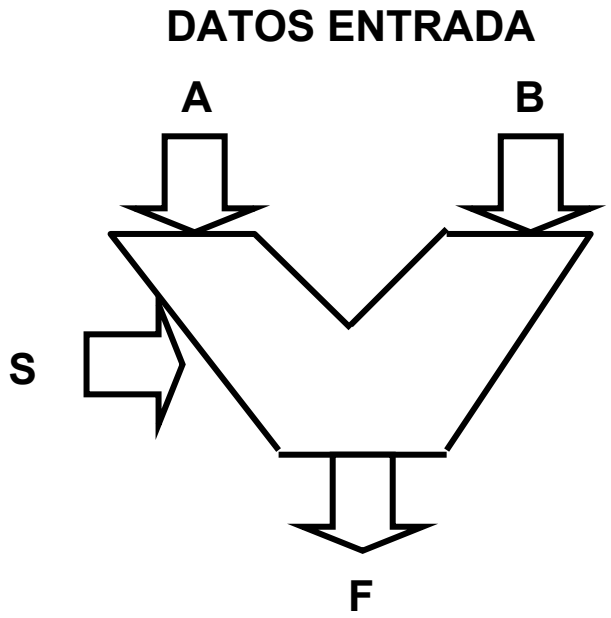
LA SENTENCIA CASE ES DE SELECCION Y PERMITE SELECCIONAR UNA ENTRE VARIAS ALTERNATIVAS - VA JUNTO A LA SENTENCIA WHEN

```
process (<entradas separadas por comas>)
begin
   Instrucciones;
end process;
```

ALU - UNIDAD LOGICA ARITMETICA

LA ALU ES UN DISPOSITIVO QUE EJECUTA OPERACIONES LOGICAS Y ARITMETICAS. A SABER:

LOGICAS: AND, OR, NAND, NOR, XOR, NEGACION.
ARITMETICAS: SUMA, RESTA, COMPARACION DE MAGNITUD, SHIFT.



A TRAVES DE APROPIADAS SEÑALES DE CONTROL <S> SE PUEDEN SELECCIONAR LAS DIFERENTES FUNCIONES LOGICAS Y ARITMETICAS

MODELO VHDL - ALU

En la tabla que sigue se especifica el funcionamiento de la ALU 74381. Tiene 2 entrada de datos de 4 bits c/u denominadas A y B, una entrada de selección *s* de 3 bits y una salida *F* de 4 bits. En la tabla el signo + indica adición aritmética y el signo - significa sustracción aritmética.

OPER	ENT S_2 S_1 S_0	SALIDA F
Clear	0 0 0	0 0 0 0
B - A	0 0 1	B - A
A - B	0 1 0	A - B
ADD	0 1 1	A + B
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_ logic _unsigned.all ;

ENTITY alu IS
    PORT ( s     : IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
           A, B : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             F     : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END alu ;
```

MODELO VHDL - ALU

```
•ARCHITECTURE Behavior OF alu IS
•BEGIN
•       PROCESS ( s, A, B )    -- LISTA DE SENSIBILIDAD
•       BEGIN
•               CASE s IS
•                       WHEN “000” =>
•                               F <= “0000” ;
•                       WHEN “001” =>
•                               F <= B – A ;
•                       WHEN “010” =>
•                               F <= A – B ;
•                       WHEN “011” =>
•                               F <= A + B ;
•                        WHEN “100” =>
•                               F <= A XOR B ;
•                        WHEN “101” =>
•                               F <= A OR B ;
•                        WHEN “110” =>
•                               F <= A AND B ;
•                        WHEN OTHERS =>
•                               F <= “1111” ;
•               END CASE ;
•       END PROCESS ;
•END Behavior ;
```

OPER	ENT	SALIDA
	S_2 S_1 S_0	F
Clear	0 0 0	0 0 0 0
B - A	0 0 1	B - A
A - B	0 1 0	A - B
ADD	0 1 1	A + B
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1