

5. Representación numérica y circuitos de aritmética

Agradecimientos a Guido Muñoz

angel@uc.cl

IEE2713 Sistemas Digitales



5.1 – 5.2.3

Representación de números sin signo

angel@uc.cl

IEE2713 Sistemas Digitales

5.1.1 Enteros sin signo

- Un número N_B puede ser representado con n digitos d_i en base B, de manera que:

$$N_B = d_{n-1} \times B^{n-1} + d_{n-2} \times B^{n-2} + \cdots + d_1 \times B^1 + d_0 \times B^0$$

- En la vida cotidiana, esta base corresponde a 10 (números decimales). En los computadores, se usa la base 2 (números binarios):

$$(1001)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (9)_{10}$$

- Un grupo de 4 dígitos binarios es un nibble – ej: 1001
- Un grupo de 4 dígitos binarios es un byte – ej: 10110100

¿Qué es el LSB y el MSB?

5.1.2 Conversión entre decimal y binario

- Binario a decimal: ponderamos cada bit por su peso

$$D = d_{n-1}d_{n-2}\cdots d_1 d_0$$

$$V(D) = d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + \cdots + d_1 \times 10^1 + d_0 \times 10^0$$

- **¿Cómo lo hacemos con binarios no enteros?**

5.1.2 Conversión entre decimal y binario

- Decimal a binario: Dividimos sucesivamente por 2

Convert $(857)_{10}$

			Remainder	
$857 \div 2$	=	428	1	LSB
$428 \div 2$	=	214	0	
$214 \div 2$	=	107	0	
$107 \div 2$	=	53	1	
$53 \div 2$	=	26	1	
$26 \div 2$	=	13	0	
$13 \div 2$	=	6	1	
$6 \div 2$	=	3	0	
$3 \div 2$	=	1	1	
$1 \div 2$	=	0	1	MSB

Result is $(1101011001)_2$

$$\frac{V}{2} = b_{n-1} \times 2^{n-2} + \cdots + b_2 \times 2^1 + b_1 + \frac{b_0}{2}$$

¿Cómo lo hacemos con decimales no enteros?

5.1.3 Representaciones octal y hexadecimal

Decimal	Binary	Octal	Hexadecimal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12

- La base de un número **no está limitada** solo a 2 y 10. Octal (base 8) y hexadecimal (base 16) son ocupadas en ciertos casos.
- En octal se ocupan dígitos del 0 al 7. En hexadecimal se ocupan desde el 0 al 9, y a continuación se ocupan letras desde A(10) hasta F(15).

Conversión binario-octal-hex

- Conversiones entre binario y octal son triviales, 3 dígitos binarios corresponden a 1 dígito octal: $(010111011)_2 = (273)_{10}$

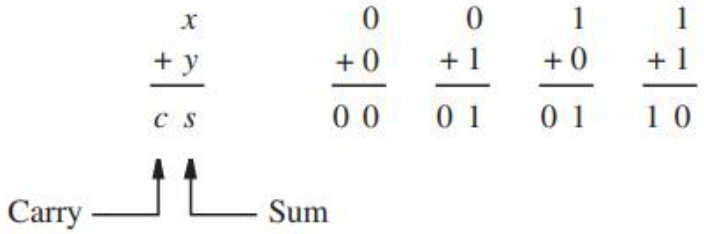
$$\underbrace{0\,1\,0}_{2} \qquad \underbrace{1\,1\,1}_{7} \qquad \underbrace{0\,1\,1}_{3}$$

Zero padding en caso que quedemos con menos bits

- Conversiones entre binario y hexadecimal son triviales, 4 dígitos binarios corresponden a 1 dígito hexadecimal: $(1010111100100101)_2 = (AF25)_{16}$

$$\underbrace{1\,0\,1\,0}_{A} \qquad \underbrace{1\,1\,1\,1}_{F} \qquad \underbrace{0\,0\,1\,0}_{2} \qquad \underbrace{0\,1\,0\,1}_{5}$$

5.2 Suma de números sin signo



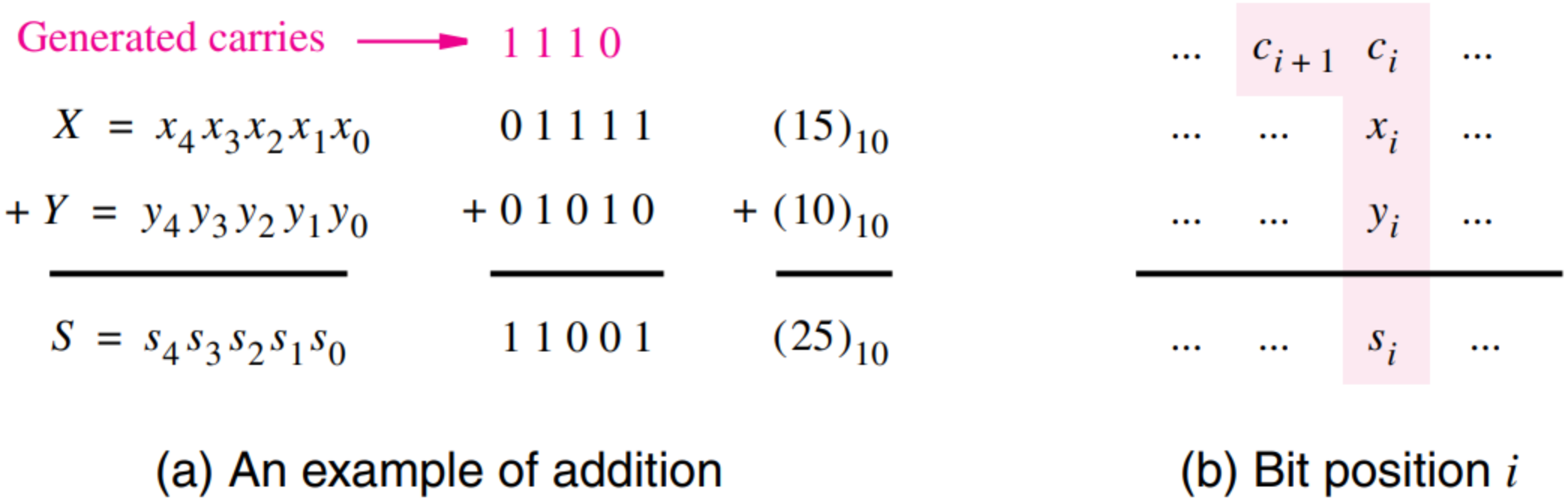
(a) The four possible cases

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table

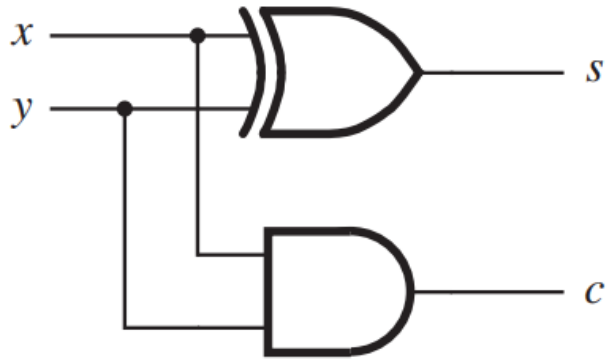
- En números binarios la suma es realizada bajo el mismo principio que en los decimales, pero los resultados sólo pueden ser 0 o 1.
- Se requieren 2 bits para representar el resultado. El bit de más a la derecha es llamado suma *(sum),* el bit de más a la izquierda es llamado *carry.*
- **¿A qué funciones lógicas corresponden c y s?**

Ejemplo de suma

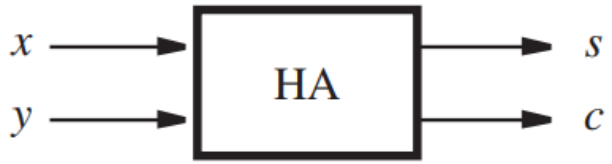


Half Adder

- Un circuito capaz de sumar 2 bits de entrada es llamado un **Half Adder** (HA). Corresponde a un XOR para la suma y un AND para el carry:



(c) Circuit



(d) Graphical symbol

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

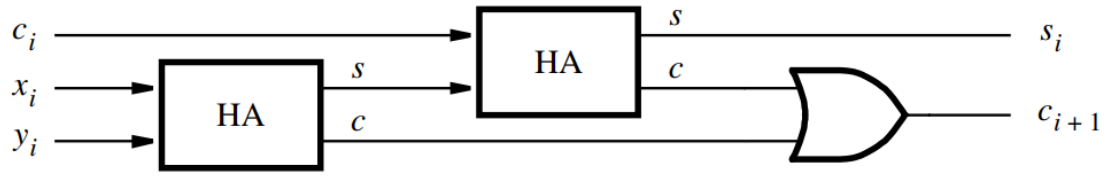
Figure 3.1 Half-adder.

Tratemos de diseñar un sumador de palabras de 5 bits

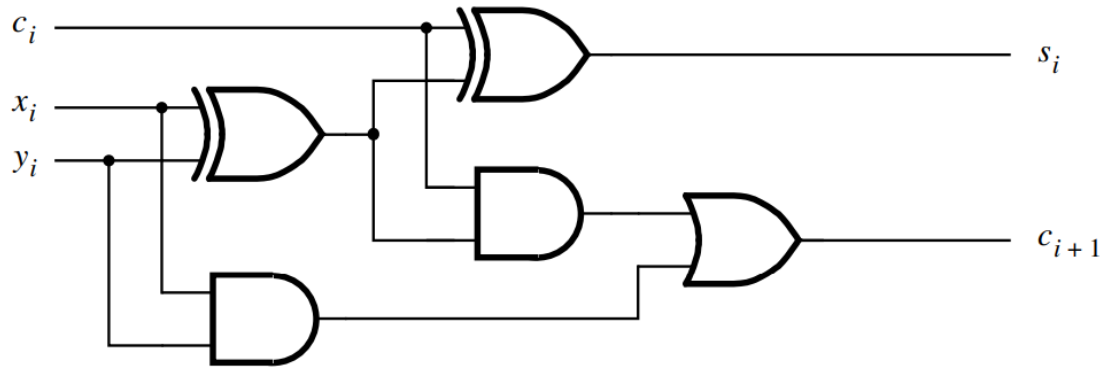
Comencemos con la tabla de verdad:

La solución: Full Adder (concatenable)

- Se le llama **Full Adder** (FA) a un circuito capaz de sumar 2 bits de entrada y, además, incorporar un carry de entrada:



(a) Block diagram



(b) Detailed diagram

A	B	Cin	Sum	Cout
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

(en el fondo, un Full Adder suma 3 bits de entrada)

5.2.2 Ripple-carry adder

- Si necesitamos sumar números de más de un bit, podemos construir un sumador adecuado usando Full Adders:

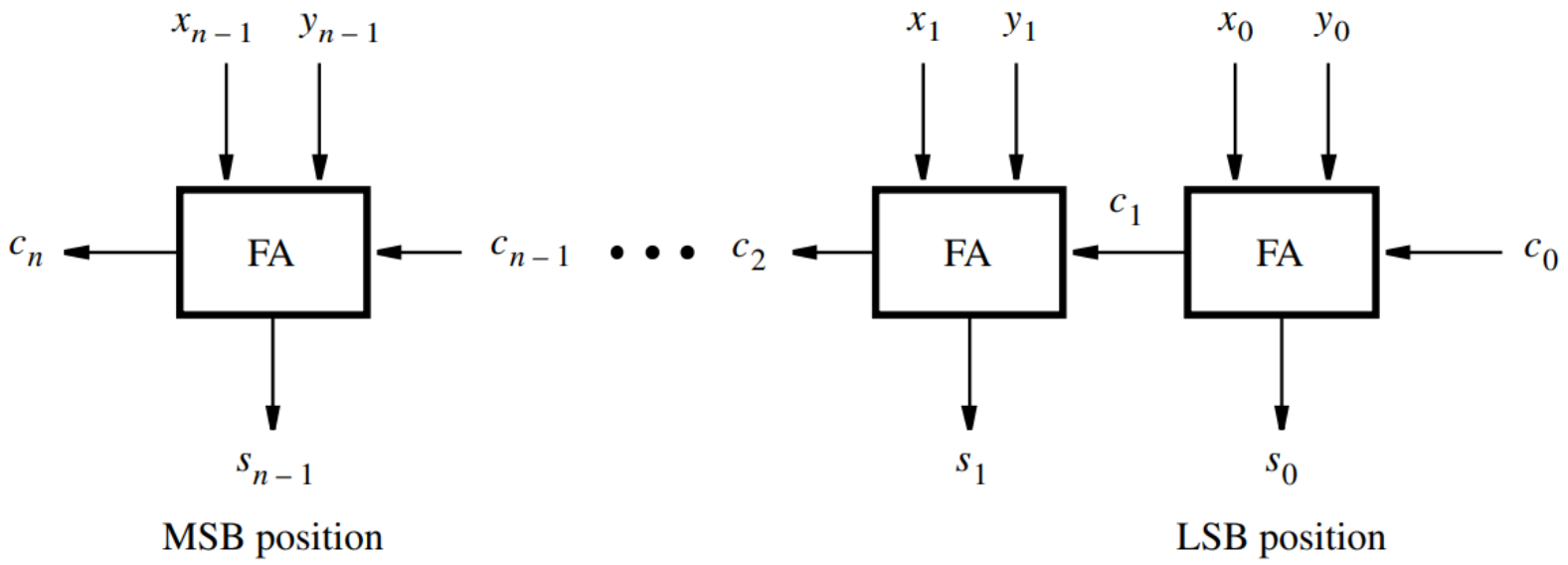


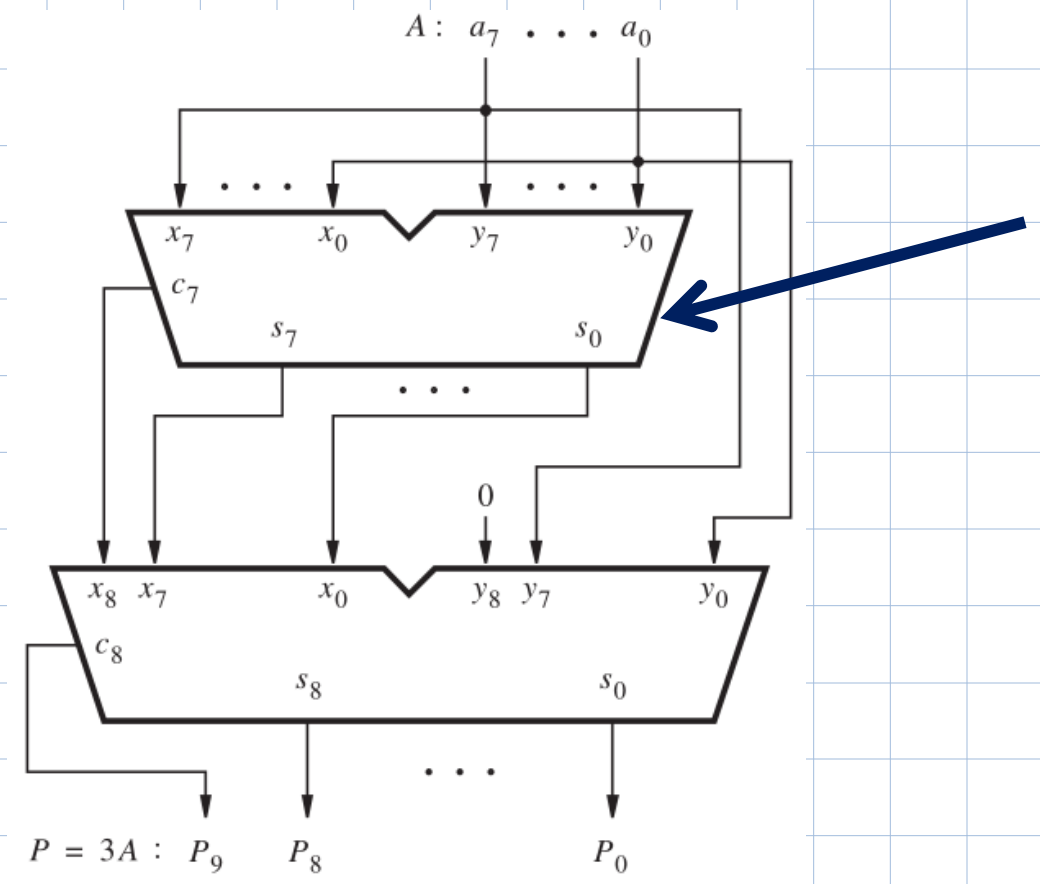
Figure 3.5 An n-bit ripple-carry adder.

La reserva o carry se propaga hacia la izquierda "en olitas" (ripple) entre cada FA. En sumadores de muchos bits (32 o 64), el retardo del carry en llegar al MSB puede ser inaceptable.

5.2.3 Ejemplo de diseño

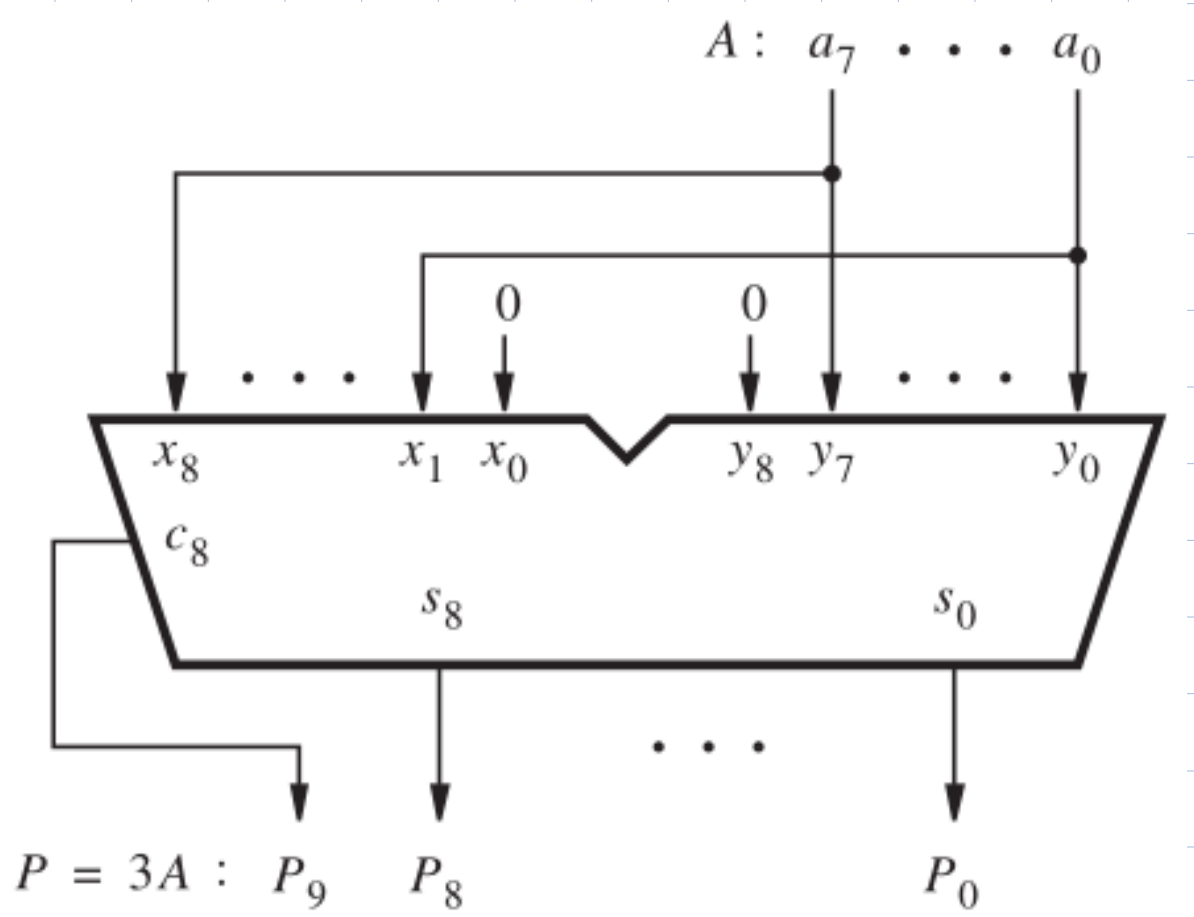
- Hagamos un circuito que multiplique por 3:

  P = 3A



Este bloque simboliza un sumador

Repitamos haciendo P=2A+A



Esta implementación es mucho más eficiente



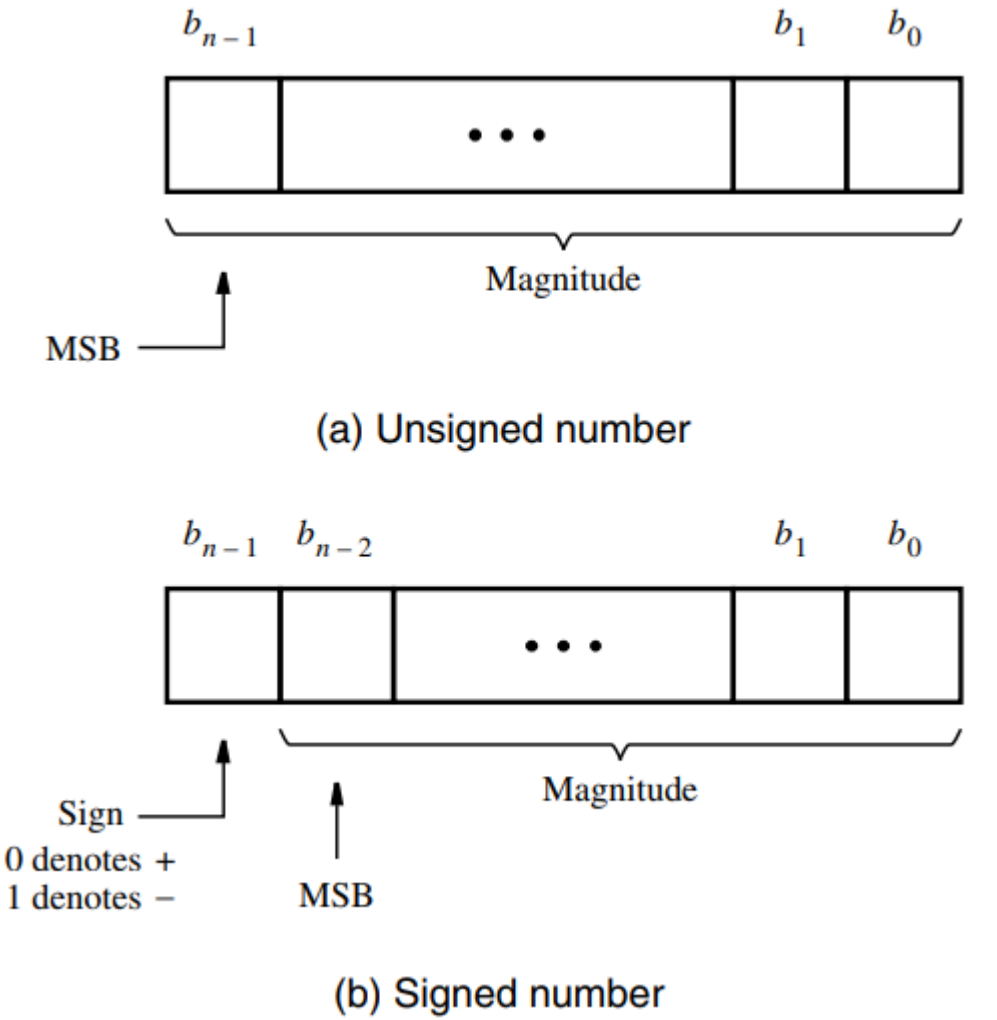
5.3 – 5.3.6

Números con signo

angel@uc.cl

IEE2713 Sistemas Digitales

5.3 Números con signo



- Para representar bits con signo, se requiere un bit adicional.
  - El bit de signo se activa para números negativos
- Nótese el cambio en la posición del bit mas significativo.

5.3.1 Números negativos

- Para representar números negativos existen 3 formas:

1. Representación de signo y magnitud *(signed-and-magnitude representation).*
El bit de signo está en verde, mientras que el MSB en rojo.

$+7_{10}=0111$	$-7_{10}=1111$

¿Cómo representamos el 0?

Representación en complemento de 1

- Otra forma para representar los negativos es a través del complemento de 1.

Complemento de -1 de -7

$n = 5;\ P = 00111$

$K = (2^5 - 1) - 00111 = 11111 - 00111 = 11000$

$31_{10} = 11111_2$

Es fácil obtener un número negativo usando complemento de 1, pero tiene desventajas al usar esta representación en circuitos

Receta:

1. Determinar el número de bits del número (n) y el número en positivo (P)
2. Calcular el complemento (K) como:

$$K = (2^n - 1) - P$$

O bien, intercambiar 1s y 0s del binario original.

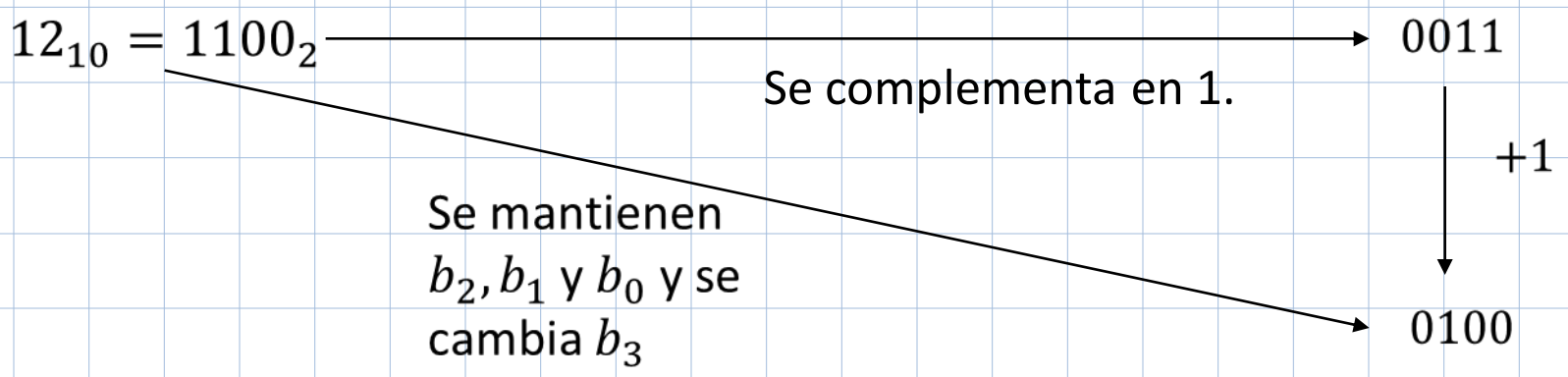
¿Cómo representamos el 0?

Representación en complemento de 2

- Existen dos formas de encontrar el complemento de 2 de un número:

1. Copiar los bits (LSB a MSB) hasta el primer 1 (inclusive), y después invertir el resto de los 1s y 0s
2. Encontrar el complemento 1 (invertir bit a bit) y sumarle 1.

Esta representación es la más usada, por lejos.



¿Qué sucede si hacemos complemento 2 de un número que ya es negativo? ¿Cómo representamos el 0?

Representación mediante signo y magnitud, complemento de 1, y 2

Table 3.2 Interpretation of four-bit signed integers.

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	−0	−7	−8
1001	−1	−6	−7
1010	−2	−5	−6
1011	−3	−4	−5
1100	−4	−3	−4
1101	−5	−2	−3
1110	−6	−1	−2
1111	−7	−0	−1

¿Cómo encontrar el valor decimal de un número binario representado en complemento de 2?

$$V(B) = (-b_{n-1} \times 2^{n-1}) + b_{n-2} \times 2^{n-2} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

¿Cuáles son los números más grandes y más pequeños que podemos representar con n bits en complemento 2?

Suma y resta

- La suma de números positivos es la misma en cualquiera de las tres representaciones de números con signo

(+5)	0 1 0 1
+ (+2)	+ 0 0 1 0
(+7)	0 1 1 1

- Hay muchas diferencias entre las tres representaciones cuando queremos sumar números negativos, o sumar números positivos con números negativos
- La resta es particularmente complicada en representación magnitud signo – por eso no usamos esta representación en computadores

Suma / resta complemento 1

- Hay 4 casos posibles: suma de positivos, suma de negativos, y suma con distinto signo (2 casos)

(+5)	0101	(–5)	1010
+ (+2)	+ 0010	+ (+2)	+ 0010
(+7)	0111	(–3)	1100

Si no hay carry en el bit de signo, el resultado es inmediato.

(+5)	0101	(–5)	1010
+ (–2)	+ 1101	+ (–2)	+ 1101
(+3)	10010	(–7)	10111
	→ 1		→ 1
	0011		1000

Cuando aparece un carry en el bit de signo, tenemos que sumarlo al resultado para obtener el valor correcto.

Esto hace que la suma / resta de números con representación complemento 1 no es muy práctica.

Suma / resta en complemento 2

- Hay 4 casos posibles: suma de positivos, suma de negativos, y suma con distinto signo (2 casos)

(+5)	0 1 0 1	(–5)	1 0 1 1
+ (+2)	+ 0 0 1 0	+ (+2)	+ 0 0 1 0
(+7)	0 1 1 1	(–3)	1 1 0 1
(+5)	0 1 0 1	(–5)	1 0 1 1
+ (–2)	+ 1 1 1 0	+ (–2)	+ 1 1 1 0
(+3)	1 0 0 1 1	(–7)	1 1 0 0 1
	ignore		ignore

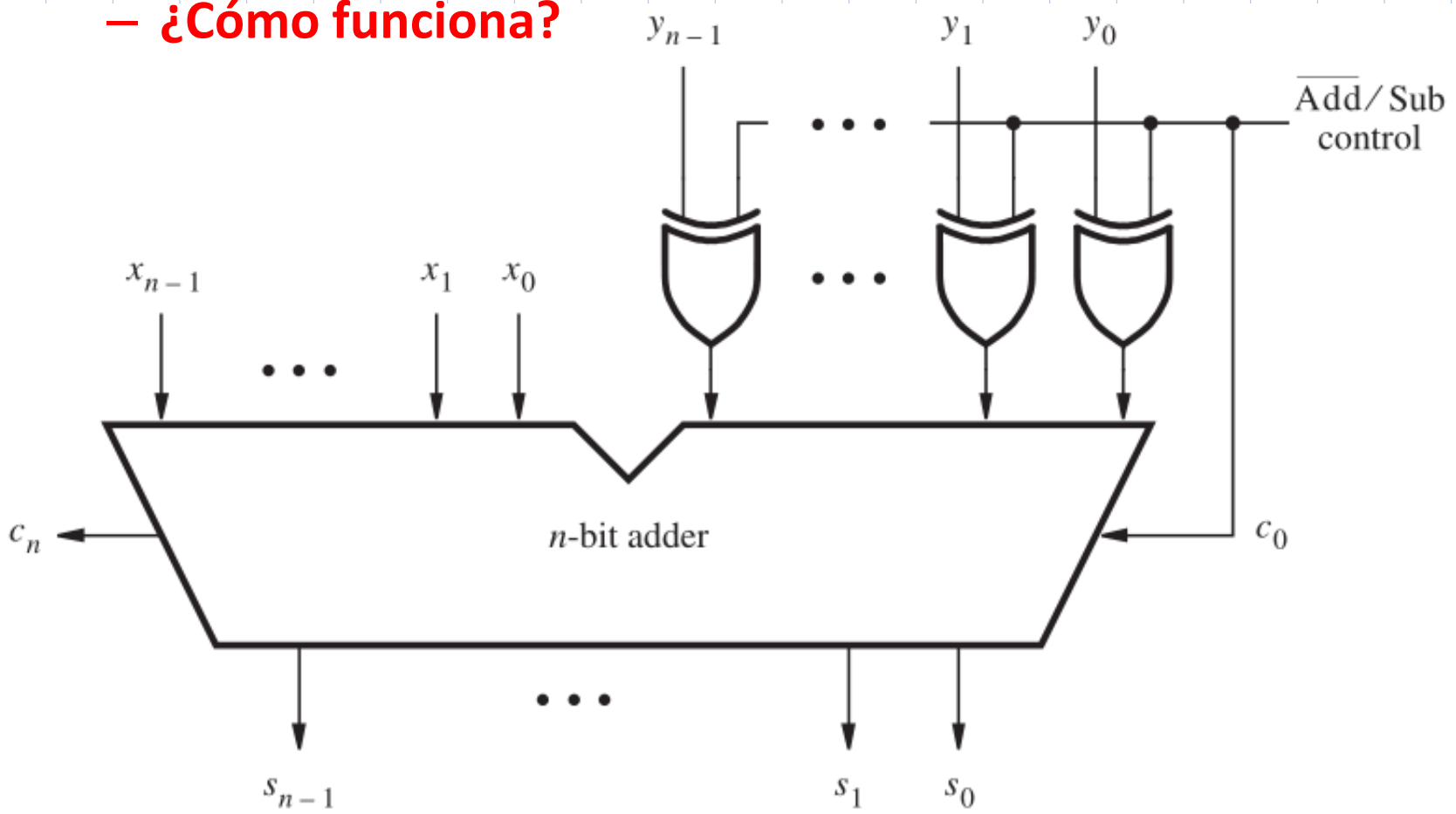
Si hay carry , hay que ignorarlo.

Para restar números, simplemente aplicamos complemento 2 al sustraendo y luego sumamos.

Entonces la suma / resta de números con representación complemento 2 es trivial y por ello es lo más usado en circuitos digitales.

5.3.3 Bloque sumador / restador complemento 2

- Este bloque puede sumar y restar…
  - **¿Cómo funciona?**



5.3.4 Esquemas complemento radix

- Para entender un poco la logica detrás de los complementos de 1 y 2 se puede pensar en como estos funcionan cuando se trabaja con números decimales.

$$74 - 36 = 74 + 100 - 100 - 36$$
$$= 74 + (100 - 36) - 100$$

$$74 - 36 = 74 + (99 + 1 - 36) - 100$$
$$= 74 + (99 - 36) + 1 - 100$$

Nótese que 74-36 requiere un "prestamo" del digito anterior. Para evitar esto se puede usar complementos sobre 36, lo que implica:

1. Restar 99-36 y luego sumar el carry, lo que ocupa complemento de 9, análogo al complemento 1
2. Restar 99 – 36 +1, y luego, de haber carry, ignorarlo, lo que ocupa complemento de 10, análogo al complemento de 2.

Para mas información visitar página 140 del texto guía en 3ra edición.

5.3.5 Overflow aritmético

- Si el resultado de una suma involucra ocupar más bits de los que se disponen, entonces ocurre un *overflow*.

(+7)	0 1 1 1	(−7)	1 0 0 1
+ (+2)	+ 0 0 1 0	+ (+2)	+ 0 0 1 0
(+9)	1 0 0 1	(−5)	1 0 1 1
	$c_4 = 0$, $c_3 = 1$		$c_4 = 0$, $c_3 = 0$
(+7)	0 1 1 1	(−7)	1 0 0 1
+ (−2)	+ 1 1 1 0	+ (−2)	+ 1 1 1 0
(+5)	1 0 1 0 1	(−9)	1 0 1 1 1
	$c_4 = 1$, $c_3 = 1$		$c_4 = 1$, $c_3 = 0$

Para detectar *overflow* debemos analizar si el MSB (c_3) y el bit signo (c_4) son distintos, en cuyo caso ocurrió un overflow .

$$\text{Overflow} = c_3\bar{c}_4 + \bar{c}_3 c_4 = c_3 \oplus c_4$$