Estructura de Computadores

Nociones Básicas

Nociones básicas

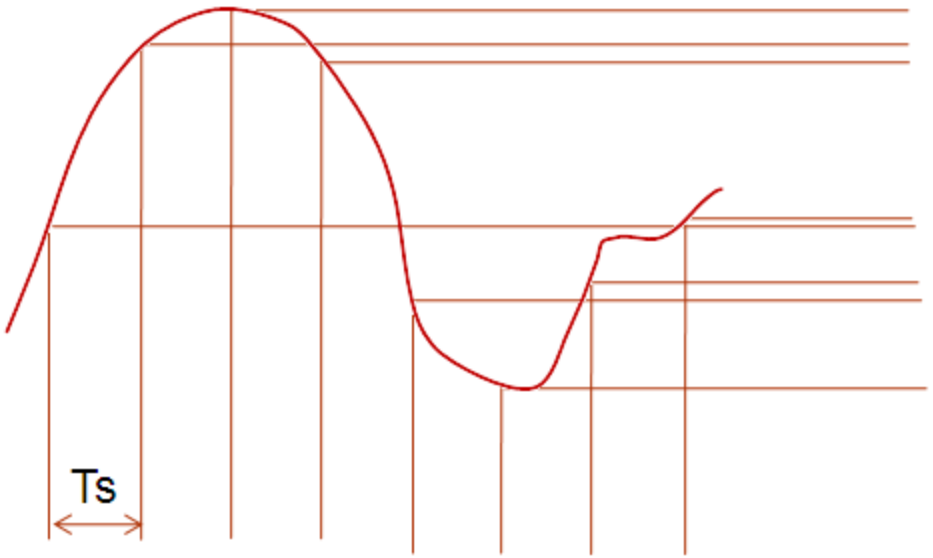
- *En este tema se repasan conceptos que aunque deberían ser conocidos se consideran importantes y se utilizan a lo largo de la asignatura, especialmente en el primer tema.*

Índice

- Digitalización
- Sistemas de numeración posicional
- Sistema binario
  - Coma fija: números positivos/negativos
  - Coma flotante
- Operaciones con números binarios
- Códigos detectores/correctores
- Almacenamiento del bit: FF, registros y contadores
- Boole, truth-table, mapas de Karnaugh
- Multiplexores y decodificadores

Digitalización

- La información analógica toma valores reales → la digitalización supone pasarla al valor discreto representable más próximo



El número de bits necesario dependerá del valor de la señal y la resolución (el error) deseada.

El lenguaje digital

- Representación de la información:
  - Sistema de numeración posicional (conversiones)
  - Sin signo/con signo
  - Coma fija/coma flotante
- Operaciones
  - Lógicas (básicas): AND, OR, XOR, NOT
  - Aritméticas: suma, resta, multiplicación, división.
- Códigos detectores y correctores
  - Distancia Hamming
  - Código Hamming

Terminología

- Bit, byte,…
  - Bit
  - Nibble: 4 bits
  - Byte: 8 bits
  - Word: su longitud dependerá de la arquitectura

- MSB, lsb
  - El bit más a la izquierda es el de mayor peso (Most Significant Bit)
  - El bit más a la derecha es el de menor peso (least significant bit)

Numeración posicional

$$Z = \sum_{i=-k}^{n-1} d_i \cdot b^i$$

donde d_i=i-ésimo multiplicador, b=base

	b	d
Binaria	2	{0,1}
Octal	8	{0,1,2,3,4,5,6,7}
Decimal	10	{0,1,2,3,4,5,6,7,8,9}
Hexadecimal	16	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Numeración posicional

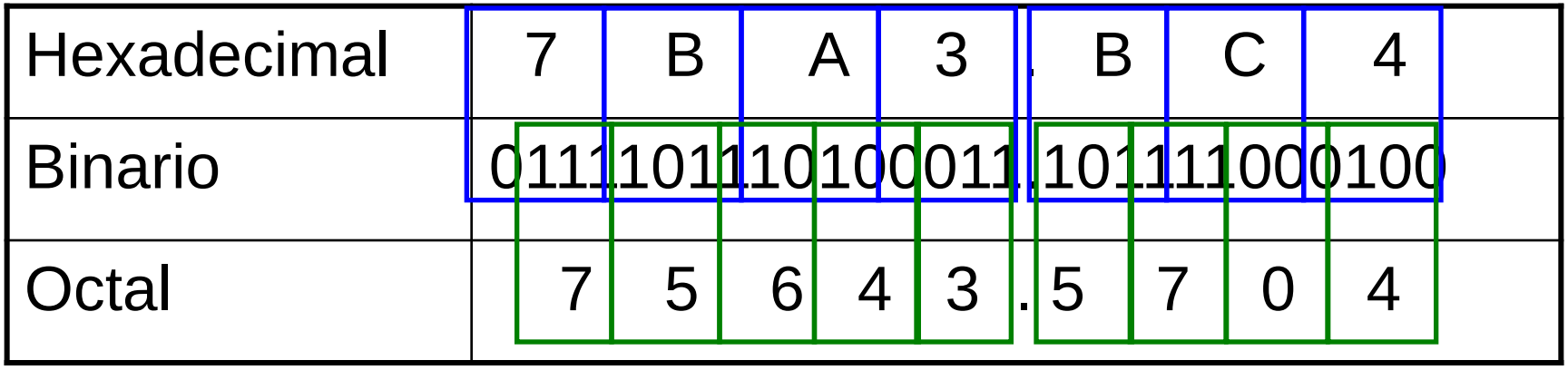
Ejemplo:

	$2001{,}125_{10}$
Binario	11111010001,001
Octal	3721,1
Decimal	2001,125
Hex	7D1,2

Nota: hexadecimal para direccionamiento (laboratorio)

Conversión entre bases (1)

Octal ⟺ Binario ⟺ Hexadecimal



Conversión entre bases (2)

Decimal ⟺ otras:

- Parte entera entre base deseada (hasta que la parte entera del dividendo sea menor que el divisor) → los restos (del último al primero) dan el número en la base deseada.
- La parte decimal se multiplica por la base (mientras la multiplicación tenga parte decimal) → las partes enteras dan el número.

Práctica

Pasa a binario los siguientes números decimales:

8_{10}=

28_{10}=

$14{,}125_{10}$=

$256{,}75_{10}$=

Convierte a decimal los siguientes números:

1010_2=

FFh=FF_{16}=

Práctica: solución

Pasa a binario los siguientes números decimales:

$8_{10}=1000_2$

$28_{10}=11100_2$

$14{,}125_{10}=1110{,}001_2$

$256{,}75_{10}=10000000{,}11_2$

Convierte a decimal los siguientes números:

$1010_2=10_{10}$

$FFh=FF_{16}=255_{10}$

Decimal Codificado en Binario

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
~~no~~ utilizados	~~1010~~ 1011 1100 1101 1110 1111

Personas → decimal

+

Máquinas → lógica binaria

=

BCD (*Binary Coded Decimal)*

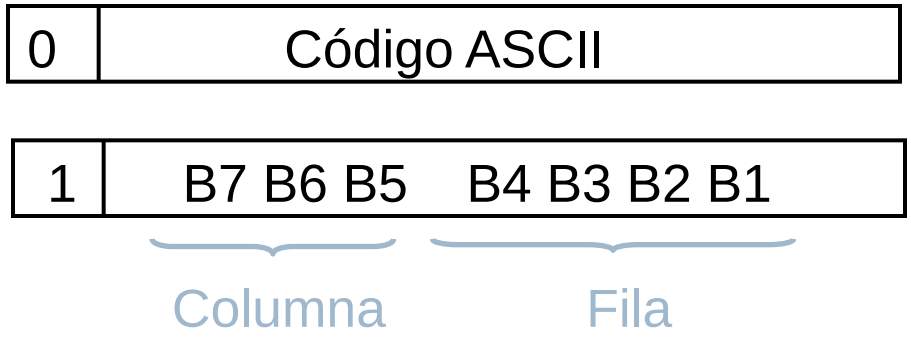
Ej: 396_{10}= 0011 1001 0110

(solo 9 de las 16 combinaciones)

Códigos alfanuméricos: ASCII (1)

ASCII: *American Standard Code for Infomation Interchange*

- 7 bit→128 caracteres
- 1 byte (MSB=1) → 128 combinaciones para caracteres especiales



Códigos alfanuméricos: ASCII (2)

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

ASCII Code Chart-Quick ref card (wikipedia, dominio público)

Números con signo (1)

¿Cómo representar con un número dado de bits números negativos?

La mitad de las combinaciones representan números positivos y la otra mitad negativos. Opciones:

- Signo y magnitud
- Complemento a uno
- Complemento a dos
- Exceso 2^{n-1}

Números con signo (2)

Signo y magnitud

- MSB indica el signo: '0' positivo, '1' negativo
- El resto representan la magnitud
- Ejemplo:
  - 0101=5
  - 1101=-5
  - 0000=0=1000=-0 (doble representación para el cero)

Números con signo (3)

Complemento a uno

- Números no negativos misma representación que los números sin signo (y que S&M)
- Se obtiene invirtiendo todos los bits (0 ⇔ 1). Es una operación bidireccional: permite obtener el número negativo a partir del positivo, pero también al revés.
- Ejemplo:
  - 0101=5
  - 1010=-5
  - 0000=0=1111=-0 (doble representación para el cero)

Números con signo (4)

Complemento a dos

- Números no negativos misma representación que los números sin signo (y que S&M y Ca1)
- Se obtiene sumando uno al complemento a uno. Se trata, al igual que el complemento a uno, de una operación bidireccional.
- Ejemplo:
  - 0101=5
  - 1010+1=1011=-5
  - 0000=0 (representación única para el cero→ se puede representar un número más que con las otras representaciones)

Números con signo (5)

- Nota:
  - Aunque el complemento a 1 y a 2, como operación, puede aplicarse a cualquier conjunto de bits (son bidireccionales); en cuanto a la representación de números con signo, al igual que en S&M, el MSB indica el signo (‘0’ positivo, ‘1’ negativo).

Números con signo (6)

Exceso a $2^{n-1}-1$ (2^{n-1})

- Todos los números tienen $2^{n-1}-1$ en exceso
- Restando $2^{n-1}-1$ se obtiene el número
- Ejemplo:
  - $1111=15-(2^{4-1}-1)=15-(2^3-1)=15-7=8$
  - $0000=0-(2^{4-1}-1)=-7$
  - 0111=0 (representación única para el cero→ se puede representar un número más que con Ca1 o S&M)

Números con signo (7)

- Rango representable con n bits:
  - Binarios positivos: $0 \leq N° \leq 2^n-1$
  - Signo y magnitud: $-(2^{n-1}-1) \leq N° \leq 2^{n-1}-1$
  - Complemento a 1: $-(2^{n-1}-1) \leq N° \leq 2^{n-1}-1$
  - Complemento a 2: $-2^{n-1} \leq N° \leq 2^{n-1}-1$
  - Exceso 2^{n-1} : $-2^{n-1} \leq N° \leq 2^{n-1}-1$
  - Exceso $2^{n-1}-1$: $-(2^{n-1}-1) \leq N° \leq 2^{n-1}$

Práctica

- Rellena la tabla con el rango representable.

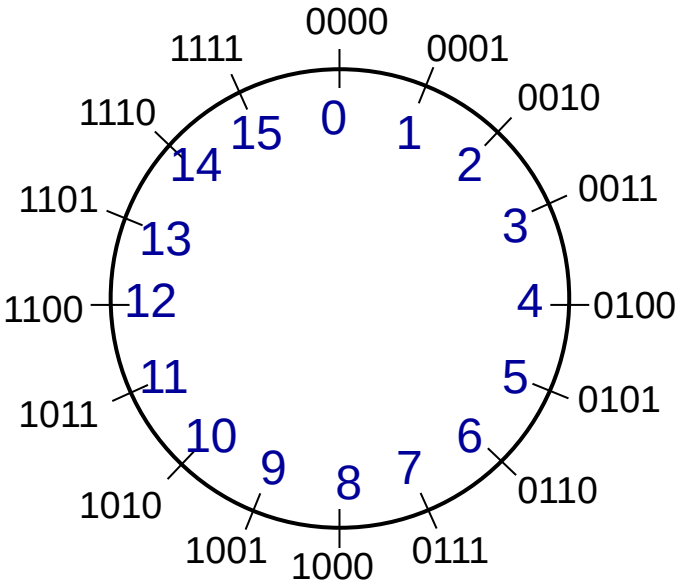
	4 bits	8 bits
Sin signo		
Signo y magnitud		
Complemento a 1		
Complemento a 2		
Exceso (7, 127)		

Práctica: solución

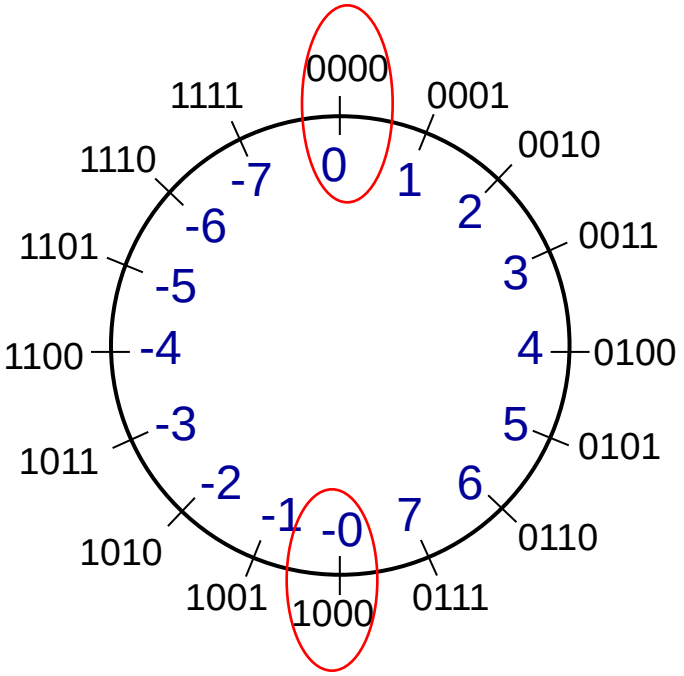
- Rellena la tabla con el rango representable.

	4 bits	8 bits
Sin signo	De 0 a 15	De 0 a 255
Signo y magnitud	De -7 a 7	De -127 a 127
Complemento a 1	De -7 a 7	De -127 a 127
Complemento a 2	De -8 a 7	De -128 a 127
Exceso (7, 127)	De -7 a 8	De -127 a 128

Números con signo (VIII)

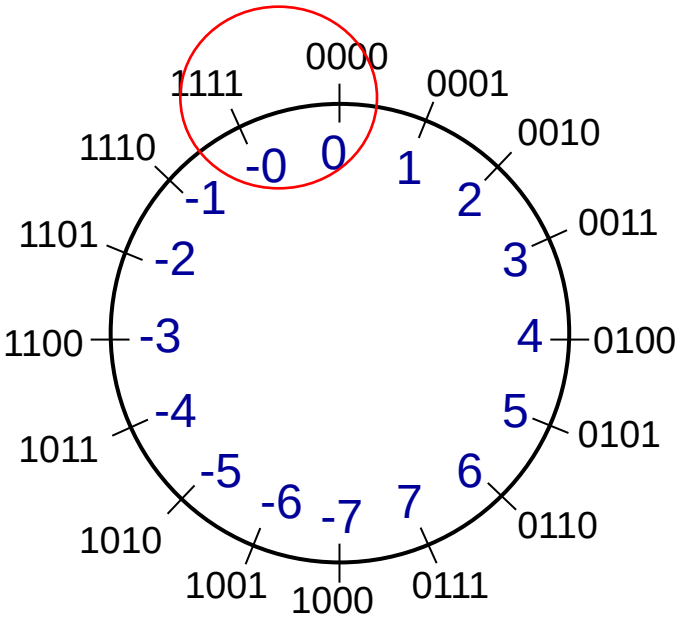


Números binarios positivos



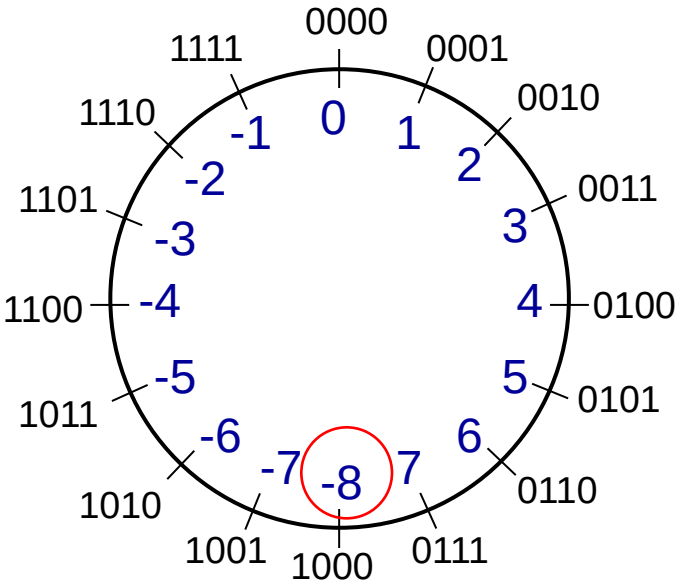
Signo y magnitud

Números con signo (IX)



Complemento a 1

$A^{(1)} = 2^N-1-|A|$



Complemento a 2

$A^{(2)} = 2^N-|A| = A^{(1)}+1$

Números con signo (X)

Decimal	Signo y magnitud	Complemento a 1	Complemento a 2	Exceso 2^{n-1}-1
-8	------	------	1000	------
-7	1111	1000	1001	0000
-6	1110	1001	1010	0001
-5	1101	1010	1011	0010
-4	1100	1011	1100	0011
-3	1011	1100	1101	0100
-2	1010	1101	1110	0101
-1	1001	1110	1111	0110
-0	1000	1111	------	------
0	0000	0000	0000	0111
1	0001	0001	0001	1000
2	0010	0010	0010	1001
3	0011	0011	0011	1010
4	0100	0100	0100	1011
5	0101	0101	0101	1100
6	0110	0110	0110	1101
7	0111	0111	0111	1110
8	------	------	------	1111

Notación científica: coma flotante (1)

$$Z = f \cdot 10^{e}$$

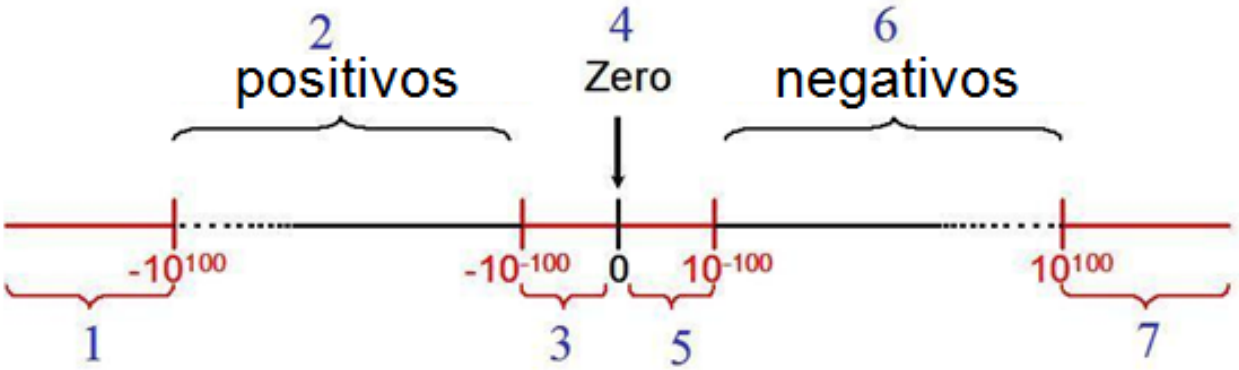
f: fracción o mantisa → precisión
e: exponente → rango, magnitud

Coma flotante: versión para los computadores

- FL (precisión doble, 64 bit)
- FS (precisión simple, 32 bit)

- **ANSI/IEEE Std. 754 (1985)**
  - Exponente (8 bit): exceso 127 para números normalizados
  - Todos '0' y todos '1' en el exponente reservados

Notación científica: coma flotante (2)



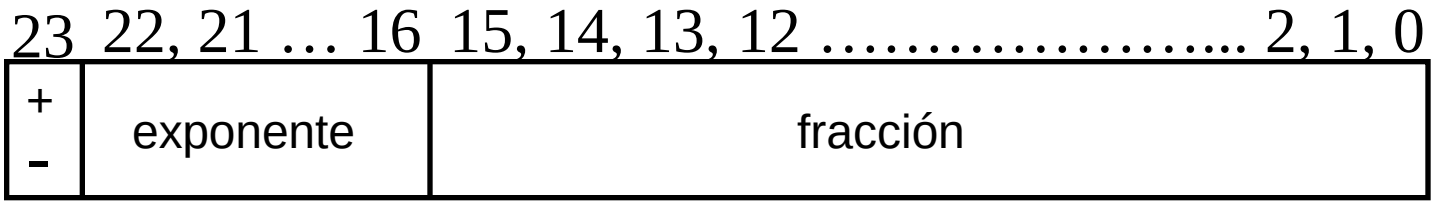
- Las zonas 1,3,5 y 7 no pueden representarse.
- Si el resultado de una operación aritmética
  - cae en las regiones 1 o 7 → *error por desbordamiento (negativo/positivo)*
  - cae en las regiones 3 o 5 → *sub-desbordamiento* (≈0).
  - no es un número discreto → *redondeo* Ej: $0{,}1 \cdot 10^3/3 \approx 0{,}333 \cdot 10^2$

Notación científica: coma flotante (3)

- Variante de la notación científica con base 2, 8 o 16.
- Cuando el dígito inmediatamente a la derecha de la coma no es un '0', se dice que el número está *normalizado*.
- Si hay ceros a la derecha de la coma, se desplazan los dígitos hacia la izquierda decrementando el exponente (se normaliza la fracción sin modificar el valor).
- *Como en binario, cualquier dígito que no sea '0' será '1', en el Std. 754 (más adelante) el primer '1' después de la coma de los números normalizados se pasa a la izquierda de la coma. De este modo el número normalizado queda como 1,XX…X. Se recupera sumando un 1 a la fracción.*

Notación científica: coma flotante (4)

Ejemplo:



Base 2, exponente en exceso 63.

No normalizado:

$01010011.0000000000011011 = 2^{20} \cdot (2^{-12}+2^{-13}+2^{-15}+2^{-16}) = 432$

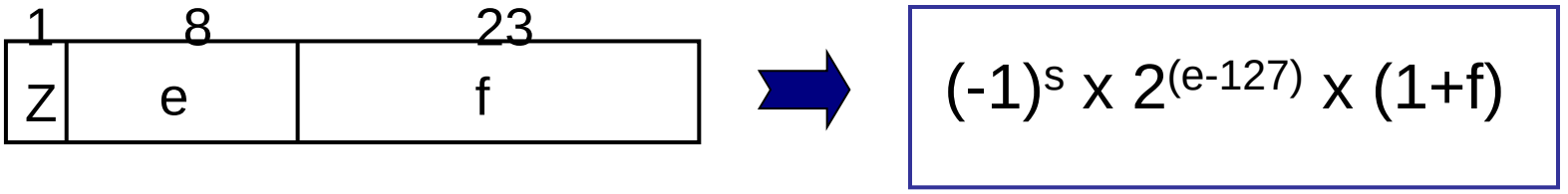
Normalizado:

$01001000.1101100000000000 = 2^{9} \cdot (2^{-1}+2^{-2}+2^{-4}+2^{-5}) = 432$

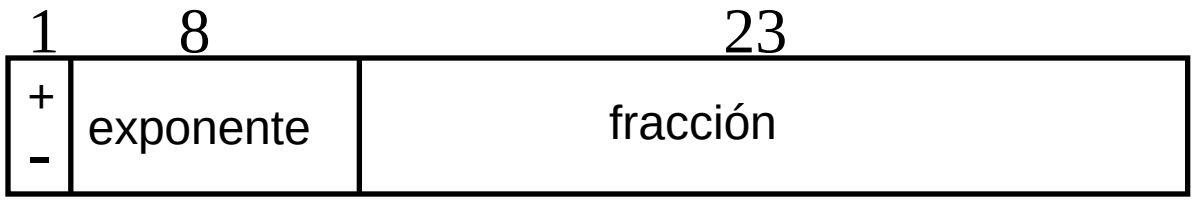
Notación científica: IEEE 754 (1)

ANSI/IEEE Std. 754 (1985)

'1' implícito en la fracción en modo normalizado (ahorro de un dígito) →



a)



b)

a) Precisión sinple *b) Precisión doble*

Notación científica: IEEE 754 (2)

• Fracción y exponente para un número normalizado y significado de los casos reservados:

	±		
Normalizado	±	0<exp.<Máx	Cualquier grupo de bits
No normalizado	±	000…00	Grupo bits diferente de cero
Cero	±	000…00	000…00
Infinito	±	111…11	000…00
NaN (no nº)	±	111…11	Grupo bits diferente de cero

Práctica

• Reescribe en notación científica, siguiendo el Std. 754 notación simple, los siguientes números (el resultado de 32 bits se representa en hexadecimal):

•100_{10}=$42C80000_{16}$=42C80000h

•$2,1_{10}$ =40066666h

•$-12,75_{10}$ =C14C0000h

•$-34,125_{10}$ =C2088000h

Operaciones binarias

- Operaciones lógicas:
  - NOT
  - AND
  - OR
  - XOR

- Operaciones aritméticas:
  - Suma con números sin signo (A:0111, B:0100)
  - Resta → suma con números en (Ca1 o) Ca2

Códigos detectores/correctores (1)

- Bit de paridad: a los bit de información se les añade uno de paridad que tomará el valor necesario ('0' o '1') de forma que el conjunto total de '1's sea par (paridad par) o impar (paridad impar). Conociendo la paridad y comprobando el número de '1's pueden detectarse errores de un bit.
- *Si tenemos 7 bits de información y uno de paridad ¿cuántas combinaciones son posibles?*
- Una combinación imposible indica que se ha dado un error en algún bit, pero no podemos saber en cuál.

Códigos detectores/correctores (2)

- La adición de más bits de paridad da lugar a más palabras código, pero sólo unas serán válidas.
  - *m* bits de datos
  - *r* bits de comprobación

  } palabra código de $n=m+r$ bits de las 2^n palabras código 2^m válidas

- El número de bits en que se diferencian dos palabras código es la **distancia Hamming**. La menor distancia Hamming entre dos palabras código válidas es la distancia Hamming del código.

Códigos detectores/correctores (3)

- d errores de un bit
  - Podrán detectarse si la distancia Hamming del código es ≥ (d+1)
  - Podrá corregirse si la distancia Hamming del código es ≥ (2d+1)
- Para poder corregir cualquier error de un bit en una palabra de m bits se necesitan r bits de comprobación / $(m+r) \leq 2^r-1$

Códigos detectores/correctores (4)

Código Hamming (paridad par):

- en las posiciones potencia de 2 hay bits de paridad
- se numeran desde la izquierda y empezando por el 1
- bit de paridad b_j =XOR de los bits que tengan un '1' en la posición 2^j
- el dato b lo comprueban b_1, b_2,…b_j / $b_1+b_2+\ldots+b_j=b$

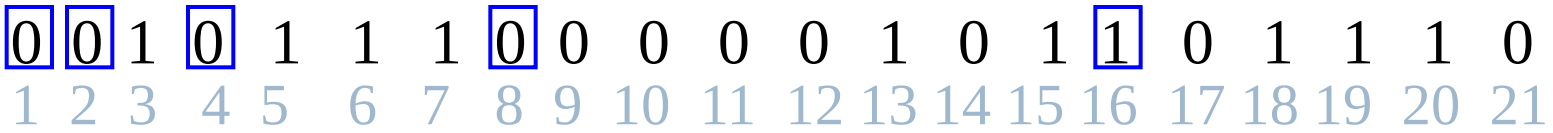
- Comprobación: cada bit de comprobación se calcula como la XOR de los bits necesarios para calcular el bit de paridad y el mismo bit. De forma que si no da '0' es que ha sucedido un error.
- La posición del bit con error nos lo darán los bits de comprobación (ci), cada uno con su peso.

Códigos detectores/correctores (5)

- *Ejemplo*

Dato: 1111000010101110

M=16 → r=5



• *Conocida la posición con error, corregir solo implica invertir el bit con error (con la señal de error y una xor)*

Códigos detectores/correctores (6)

Práctica:

- *Hamming (en transmisión): dato 0101*
- *Hamming (en recepción): 0110101*
- *Hamming (en recepción): 0010101*

Códigos detectores/correctores (6)

Práctica: Solución

- *Hamming (en transmisión): 0101*

  → *0100101*

- *Hamming (en recepción): 0110101*

  C4 C2 C1=011 → hay error en el bit 3 →

  el dato no era 1101 sino 0101

- *Hamming (en recepción): 0010101*

  C4 C2 C1=001 → hay error en el bit 1 →

  es un bit de paridad, el dato es correcto 1101

Almacenamiento de la información (1)

- Flip-flops

RS

R	S	Qt+1
0	0	Qt
0	1	1
1	0	0
1	1	No per-mitido

JK

J	K	Qt+1
0	0	Qt
0	1	0
1	0	1
1	1	/Qt

T

T	Qt+1
0	Qt
1	/Qt

D

D	Qt+1
0	0
1	1

Almacenamiento de la información (2)

Los biestables pueden almacenar un único bit. Para poder almacenar combinaciones de bits tendremos que agruparlos (presentaciones).

- Registros
  - Almacenamiento
  - Desplazamiento

Boole, Truth-Table, mapas de Karnaugh (1)

- Postulados del álgebra de **Boole**
  1. Elementos de identidad
  2. Elemento dominante
  3. Propiedad conmutativa
  4. Propiedad distributiva
  5. Axiomas del complemento
  6. Teorema de idempotencia
  7. Ley involutiva
  8. Teorema de Absorción
- Leyes de **De Morgan**

Boole, Truth-Table, mapas de Karnaugh (2)

- Funciones Booleanas

  Ej.: Diseña el sistema que active la alarma (A=1) del coche cuando se dé alguno de los siguientes casos: a) el sensor de aceite (SA=1) indica nivel bajo, el sensor de temperatura (ST=1) indica nivel alto o las luces (L=1) están encendidas y la puerta (P=1) abierta.

  Solución: $A=\pi(M_0, M_1, M_2)$

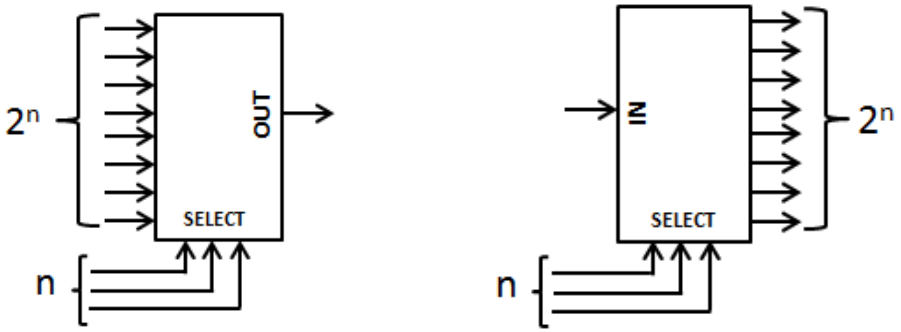
  $A=\sum(m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15})$

- Simplificación de funciones→ mapas de Karnaugh

  Solución: $A=SA+ST+L \cdot P$

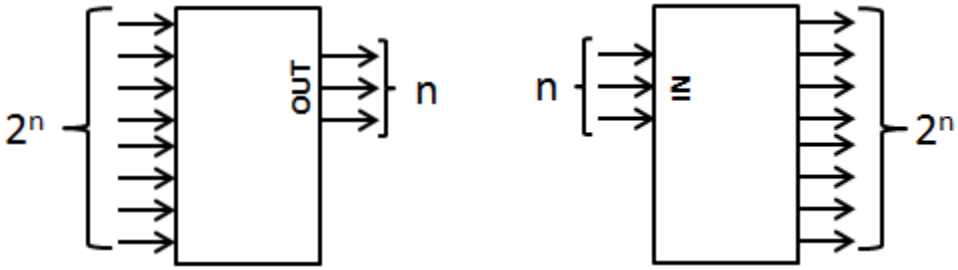
Multiplexores y codificadores

- Multiplexores / Demultiplexores



Las señales de selección indican qué entrada pasa a la salida / en qué salida se pone la entrada.

- Codificadores / Decodificadores

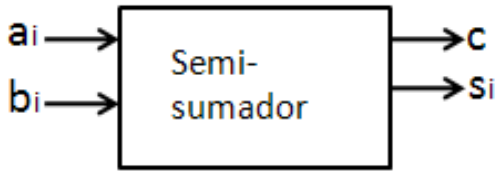


La salida (codificada en binario) indica qué entrada está activa / según el binario a la entrada, se activa la salida correspondiente.

Ej.: F=∑(m1, m2, m6). Implementa la función con un decodificador y con un multiplexor.

Sumadores

- Diseño
  - Semi-sumador
    - Admite 2 dígitos binarios en sus entradas y genera dos dígitos binarios a la salida.

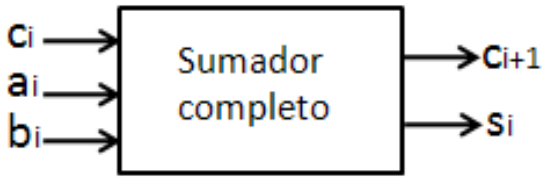


ai	bi	si	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

c = ai xor bi
si = ai and bi

Sumadores

- Sumador completo (full-adder)



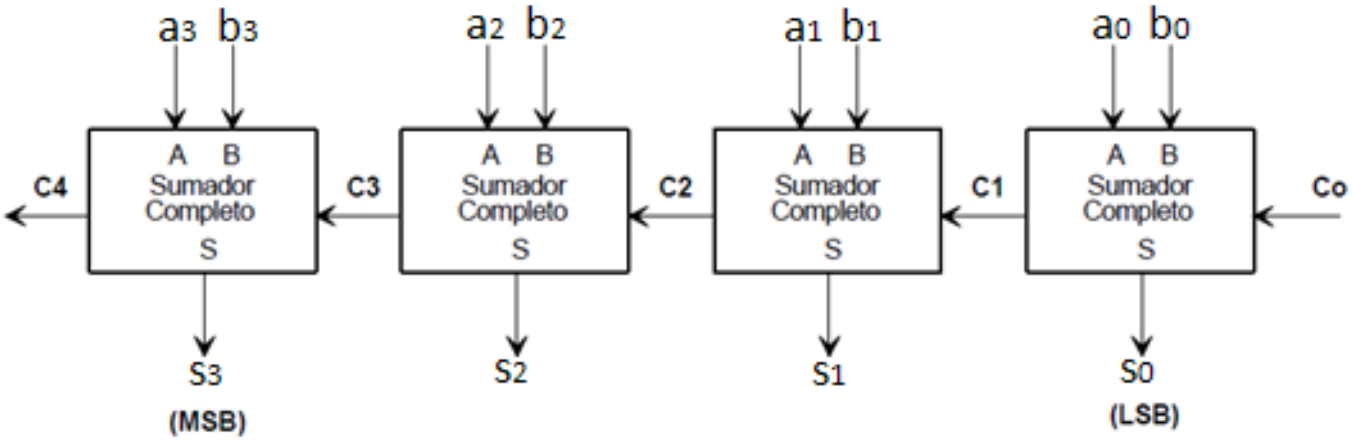
ci	ai	bi	si	ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$c_{i+1} = a_i \text{ xor } b_i \text{ xor } c_i$

$s_i = a_i \text{ and } b_i \text{ or } c_i \ (a_i \text{ xor } b_i)$

Sumadores

- Sumador binario de acarreo en cascada (ripple-carry-adder)
  - Es un sumador paralelo con llevada serie.
  - Si son números de muchos bits, la llevada en serie introduce un gran retardo.



Sustractores: Conversión de sumadores en sustractores

- La conversión de sumadores en sustractores utilizando los números en complemento a 2.
- Práctica:
  - 5+2=7 → Solución: $0101_2+0010_2=0111_2$
  - (-5)+2=(-3) → Solución: $1011_2+0010_2=1101_2$
  - 5-2=3 → Solución: $0101_2+1110_2=$~~1~~0011_2
  - (-5)-2=(-7) → Solución: $1011_2+1110_2=$~~1~~1001_2