



ARQUITECTURA DE COMPUTADORES

RESUMEN

UTN FRBA 2020
INGENIERÍA EN SISTEMAS DE LA INFORMACIÓN

Francisco Chiminelli
franciscofchiminelli@hotmail.com

UNIDAD 1: INTRODUCCIÓN A LA ARQUITECTURA DE COMPUTADORES

INFORMACIÓN Y DATO

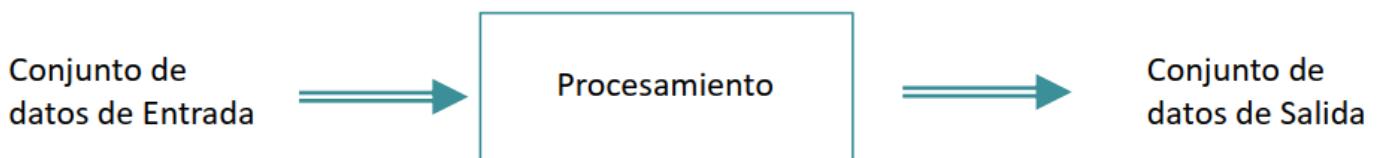
Atributo: Cualidad o característica propia que describe a una persona, una cosa o un ente.

Por ejemplo en el caso nuestro, podría ser la altura, el nombre, el color del pelo, etc. Eu un auto podría ser el modelo, el color, la potencia, etc.

Dato: Cuando a un atributo se le asigna un valor, eso es el dato.

Por ejemplo yo ahora conozco los nombres de ustedes, esos son datos, son valores de un atributo que los describe.

Proceso de datos: A partir de un conjunto de datos de entrada obtengo un conjunto de datos de salida.



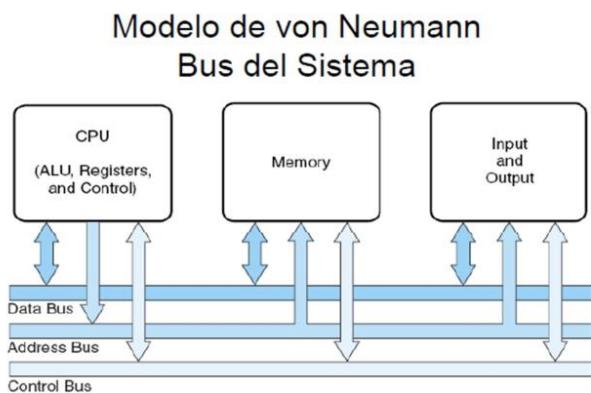
La **información** es un conjunto de **datos** seleccionados y ordenados con un propósito de brindar una información específica.

Un **dato** es la unidad mínima que compone la información. Un dato por sí mismo no es información, es el procesamiento de datos el que se transforma en información.

Computadora: dispositivo electrónico diseñado para aceptar datos de entrada, realizar operaciones sobre ellos y obtener resultados como salida.

Sus funciones básicas son: procesamiento, almacenamiento, transferencia de datos y control.

MODELO DE VON NEUMANN



3 componentes principales:

CPU:

- Unidad de Control
- Unidad aritmético lógica (ALU)
- Registros (Registros Auxiliares)

Memoria principal:

- Almacena programas y datos temporalmente.

Sistema de Entrada/Salida

UNIDAD 2: SISTEMAS DE NUMERACIÓN

SISTEMAS DE NOTACIÓN POSICIONAL (SNP)

Formados por **n** cantidad de símbolos cuya combinación representa valores diferentes.

Cada símbolo tiene un “peso” diferente según el lugar que ocupa. Es decir, el peso depende de la posición que ocupa.

El “peso” es la base elevada a la posición que ocupa dentro del número.

La suma de cada dígito multiplicado por su “peso” permite obtener el valor final del número. En SNP toda cantidad puede ser expresada como un polinomio de potencias de la base.

$$=a_n b^n + \dots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \dots + a_{-n} b^{-n}$$

a → coeficiente, símbolo del SNP

b → base del sistema

SISTEMA DECIMAL Ó BASE 10

Base de un Sistema de Numeración: Cantidad de elementos distintos que lo conforman.

Ejemplo en decimal:

1 0 2 5	Posición	Peso
	0	1
	1	10
	2	100
	3	1000

1 0 2 5

SISTEMA BINARIO Ó BASE 2

Es un sistema de base 2 lo que significa que posee 2 símbolos 0 y 1. A estos símbolos se los denomina BIT de la contracción Binary Digit

Unidades en las que trabaja la computadora:

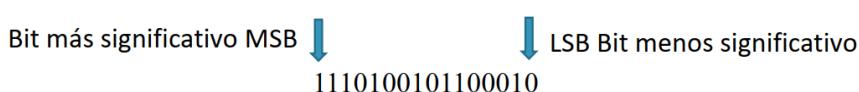
4 bits → nibble

32 bits → double Word (dWord)

8 bits → byte

64 bits → quadruple Word (qWord)

16 bits → Word (Palabra)



UNIDAD 2: SISTEMAS DE NUMERACIÓN

COMO PASAR DE BINARIO A DECIMAL

Para pasar de cualquier sistema de numeración al sistema decimal, se lo realiza a través del polinomio equivalente o polinomio de potencias de la base.

Ejemplo de binario a decimal:

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \\ \hline 1 * 2^0 & \\ 1 * 2^1 & \\ 0 * 2^2 & \\ 1 * 2^3 & \\ \hline 1 \ 1 \end{array}$$

DE OCTAL Y HEXADECIMAL A SISTEMA DECIMAL

Sistema Octal: Al ser base 8, posee esa cantidad de símbolos (0 1 2 3 4 5 6 7).

$$375_{(8)} \rightarrow 3 * 8^2 + 7 * 8^1 + 5 * 8^0 \text{ (Polinomio de potencias de la base)} = 253_{(10)}$$

Sistema Hexadecimal: Al ser base 16, posee esa cantidad de símbolos (0 1 2 3 4 5 6 7 8 9 A B C D E F)

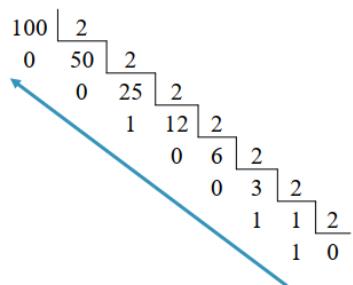
$$2B8_{(16)} \rightarrow 2 * 16^2 + 11 * 16^1 + 8 * 16^0 \text{ (Polinomio de potencias de la base)} = 696_{(10)}$$

MÉTODO DE DIVISIONES SUCESIVAS X LA BASE

Se utiliza para pasar un número entero decimal a otras bases.

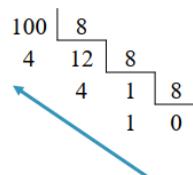
De Decimal a Binario

$$100_{(10)} \longrightarrow 1100100_{(2)}$$



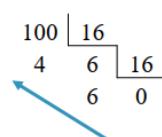
De decimal a Octal

$$100_{(10)} \longrightarrow 144_{(8)}$$



De decimal a Hexadecimal

$$100_{(10)} \longrightarrow 64_{(16)}$$



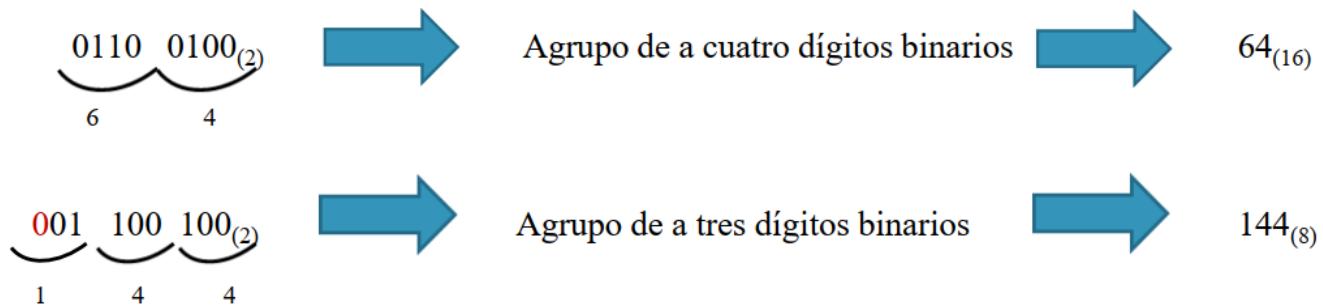
UNIDAD 2: SISTEMAS DE NUMERACIÓN

TABLA DE EQUIVALENCIAS

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10₂	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10₈	8
9	1001	11	9
10₁₀	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10₁₆

Para analizar el contenido interno de los registros de la computadora se utilizan los sistemas **octal** y **hexadecimal** ya que permiten representar su contenido de forma compacta.

Esto es posible porque 8 y 16 son potencias exactas de 2, $8=2^3$ y $16=2^4$ con lo cual se pueden convertir números del sistema **binario** al **octal** y **hexadecimal** tomando agrupaciones de 3 y 4 bits respectivamente y se lo conoce como pasaje directo entre bases.



UNIDAD 2: SISTEMAS DE NUMERACIÓN

MÉTODO DE MULTIPLICACIONES SUCESIVAS X LA BASE

Se utiliza para pasar la parte fraccionaria de un número decimal a otras bases

✓ De Decimal a Binario

$$0,2_{(10)} \longrightarrow 0,\overbrace{0011}_{(2)}$$

$$\begin{array}{l|l} 0,2 \times 2 = & 0,4 \\ 0,4 \times 2 = & 0,8 \\ 0,8 \times 2 = & 1,6 \\ 0,6 \times 2 = & 1,2 \\ 0,2 \times 2 = & 0,4 \end{array}$$

✓ De Decimal a Octal

$$0,2_{(10)} \longrightarrow 0,\overbrace{1463}_{(2)}$$

$$\begin{array}{l|l} 0,2 \times 8 = & 1,6 \\ 0,6 \times 8 = & 4,8 \\ 0,8 \times 8 = & 6,4 \\ 0,4 \times 8 = & 3,2 \\ 0,2 \times 8 = & 1,6 \end{array}$$

✓ De Decimal a Hexadecimal

$$0,2_{(10)} \longrightarrow 0,\overbrace{3}_{(16)}$$

$$\begin{array}{l|l} 0,2 \times 16 = & 3,2 \\ 0,2 \times 16 = & 3,2 \end{array}$$

Observaciones:

La parte fraccionaria siempre es 0 y se determinan los decimales tomando la parte entera del resultado de cada multiplicación en orden descendente.

Cuantos más decimales se ubican después de la coma, más preciso es el pasaje.

RESUMEN DE CONVERSIÓN ENTRE SISTEMAS NUMÉRICOS

Conversión de octal, binario o hexadecimal a decimal:

Mediante el polinomio de potencias a la base (parte entera y fraccionaria)

Conversión de decimal a hexadecimal, octal o binario:

Método de divisiones sucesivas x la base (parte entera)

Método de multiplicaciones sucesivas x la base (parte fraccionaria)

Conversión hexadecimal u octal a binario: (y viceversa)

Pasaje directo entre bases.

Conversión de octal a hexadecimal:

Se convierte primero de octal a decimal o binario y luego el valor obtenido a hexadecimal.

Conversión de hexadecimal a octal:

Se convierte primero el valor hexadecimal a decimal o binario y luego el valor obtenido a octal.

UNIDAD 2: SISTEMAS DE NUMERACIÓN

UNIDADES

Unidades básicas de información:

Medida	Unidad	Equivalencia	Notación exponencial
bit	bit	1 bit	
Byte	b	8 bits	
Kilobyte	KB	1024 bytes	2^{10}
Megabyte	MB	1024 KB	2^{20}
Gigabyte	GB	1024 MB	2^{30}
Terabyte	TB	1024 GB	2^{40}
Petabyte	PB	1024 TB	2^{50}
Exabyte	EB	1024 PB	2^{60}
Zetabyte	ZB	1024 EB	2^{70}
Yottabyte	YB	1024 ZB	2^{80}
Brontobyte	BB	1024 YB	2^{90}
Geopbyte	GeB	1024 BB	2^{100}

Submúltiplos:

Nano → 10^{-9} Atto → 10^{-18}
Pico → 10^{-12} Zepto → 10^{-21}
Femto → 10^{-15} Yocto → 10^{-24}

Tipos de datos primitivos:

Long → 64 bits Float → 32 bits
Int → 32 bits Double → 64 bits
Short → 16 bits Boolean → 32 bits
Byte → 8 bits Char → 8 bits por carácter

UNIDAD 2: SISTEMAS DE NUMERACIÓN

OPERACIONES FUNDAMENTALES EN BINARIO

Suma: la suma entre dos bits tiene 4 combinaciones posibles:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ y me llevo o acarreo } 1$$

Ejemplo →

$$\begin{array}{r} & & 111 \\ & + & 101 \\ \hline & 111 \\ & 1100 \end{array}$$

Resta: la resta se realiza con una suma, esto es sumando al minuendo el complemento a la base del sustraendo (usar circuitos sumadores para hacer restas, rehusar el mismo circuito)

$$0 - 0 = 0$$

0 - 1 = 1 le pide una unidad a la posición siguiente

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Ejemplo →

$$\begin{array}{r} & 01010 \\ - & 1101 \\ \hline & 0110 \end{array}$$

Observaciones: Para los sistemas **Octal** y **hexadecimal** se aplica el mismo método aritmético, ya que también son SNP.

COMPLEMENTO DE UN NÚMERO

El complemento (C), a la base (b), de un número (N), es lo que debo sumarle a ese número N para obtener el resultado de elevar la base a la cantidad de dígitos (formato) n del número.

Complemento a la base, a la raíz, auténtico: $C_b N = b^n - N$

Ejemplos:

$$C_{10} 7 = 10^1 - 7 = 3 \text{ para formato } n = 1$$

$$C_{10} 548 = 10^3 - 548 = 1000 - 548 = 452 \text{ para formato } n = 3$$

Complemento a la base -1 o Restringido: $C_{b-1}N = C_bN - 1$

Ejemplos en Sistema Decimal conocido en este caso como **Complemento a 9**:

$$C_{10-1} 7 = (10^1 - 1) - 7 = (10 - 1) - 7 = 9 - 7 = 2$$

$$C_{10-1} 548 = (10^3 - 1) - 548 = (1000 - 1) - 548 = 999 - 548 = 451$$

Calcular el complemento a 10 a base del Complemento a 9: $C_bN = C_{b-1}N + 1$

Complemento a 9 de un número (otro método): Se lo obtiene con lo que le falta a cada dígito para que la suma de ambos sea 9, por ejemplo el C_9 de 782 será 217. Si quiero el C_{10} para este caso será $C_9 + 1$.

UNIDAD 2: SISTEMAS DE NUMERACIÓN

COMPLEMENTO EN BINARIO

Complemento a la base, a la raíz, auténtico → C₂:

$$C_2 \ 1 = 10^1 - 1 = 1$$

$$C_2 \ 11 = 10^2 - 11 = 100 - 11 = 01$$

$$C_2 \ 110 = 10^3 - 110 = 1000 - 110 = 010$$

Complemento a la base -1 o Restringido → C₁:

$$C_{2-1} \ 1 = (101 - 1) - 1 = 1 - 1 = 0$$

$$C_{2-1} \ 11 = (102 - 1) - 11 = 11 - 11 = 00$$

$$C_{2-1} \ 110 = (103 - 1) - 110 = 111 - 110 = 001$$

Regla práctica para calcular el complemento en binario:

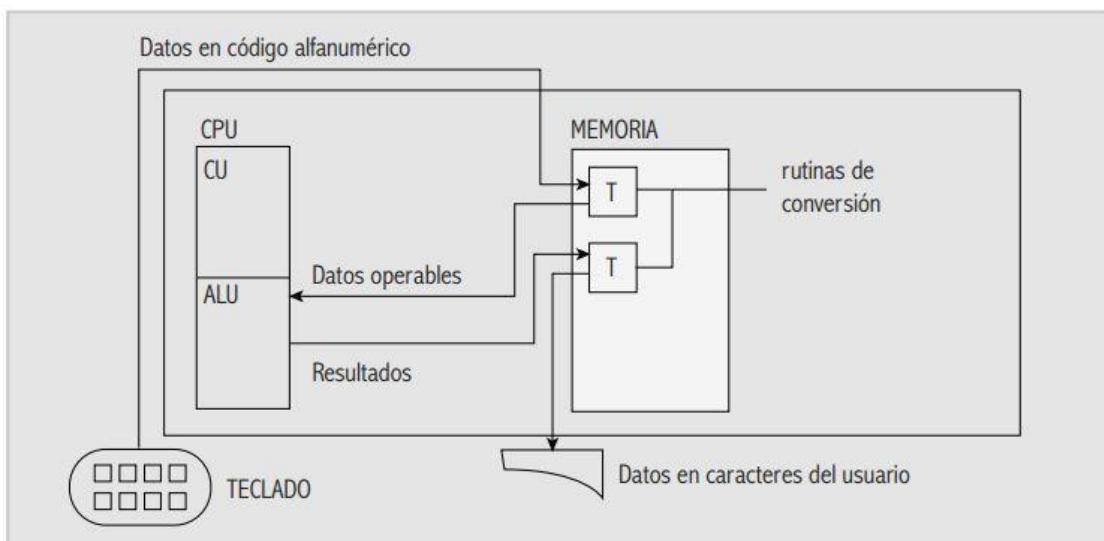
C₁: Se invierten todos los dígitos.

C₂: Se invierten todos los dígitos y se suma 1.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

FLUJO DE DATOS DENTRO DE UNA COMPUTADORA

Conversión de los datos durante el procesamiento:



T: programa de traducción (que puede ser un ensamblador, un traductor o un 2 compilador) a código de máquina

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

ALGUNAS DEFINICIONES

Código alfanumérico: es una convención que determina una combinación binaria para cada símbolo que se quiera representar.

Formato: Es la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento dentro de la computadora.

Datos primitivos en C:

Tipo	Descripción	Bytes	Rango
char	Carácter	1	-128 a 127
int	Entero con signo	2	-32768 a 32767
float	Flotante simple	4	-1.10 ⁻³⁸ a 1.10 ³⁸
double	Flotante doble	8	-1.10 ⁻³⁰⁸ a 1.10 ³⁰⁸
long	Entero largo con signo	4	-2147483648 a 2147483647
short	Entero corto con signo	2	-32768 a 32767
long double	Flotante extendido	10	-1.10 ⁻⁴⁹³² a 1.10 ⁴⁹³²
unsigned	Entero sin signo	2	0 a 65535

TIPOS DE FORMATOS DE DATOS

Representaciones alfanuméricas	ASCII UNICODE	
Representaciones decimales (BCD)	BCD puro o natural (8421) BCD exceso tres BCD 2421 o AIKEN	
Representaciones binarias	Números enteros	Coma o punto fijo sin signo (enteros-positivos) Coma o punto fijo con signo (enteros) Coma o punto fijo con complemento a la base (enteros) Coma o punto fijo con complemento restringido (enteros)
	Números reales	Coma o punto flotante (entera y fraccionaria) Coma o punto flotante con mantisa normalizada Coma o punto flotante IEEE P754 Coma o punto flotante "exceso 64"
Representaciones redundantes (detección de errores)	Códigos de paridad	Paridad vertical o a nivel carácter Paridad vertical o a nivel bloque Paridad entrelazada Código de Hamming

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

CÓDIGO ALFANUMÉRICO

Un **código alfanumérico** establece la relación necesaria para que una computadora digital, que procesa solamente dígitos binarios, interprete el lenguaje que utiliza el usuario (caracteres numéricos, alfabéticos o especiales).

Un **código** es una convención que determina una única combinación binaria para cada símbolo que se ha de representar.

Por ejemplo se ingresa el dato alfanumérico "+105", el proceso de codificación hará que éste quede registrado en la memoria, según el código empleado.

Registros de memoria	
0010 1011	+
0011 0001	1
0011 0000	0
0011 0101	5

CÓDIGOS DE REPRESENTACIÓN DE CARACTERES ALFANUMÉRICOS

String: Se trata de una unidad formada por caracteres representados según un código alfanumérico preestablecido para una computadora.

ASCII

El **"ASCII"** (Código estándar americano para intercambio de información) es uno de los códigos de representación de caracteres más usado.

El **Código ASCII de 7 bits** permite determinar 128 combinaciones posibles que representan 128 símbolos distintos.

1 carácter se representa con 7 bits → Permite $2^7 = 128$ combinaciones distintas

El **Código ASCII de 8 bits o ASCII extendido** surge debido a la necesidad de agregar caracteres.

1 carácter se representa con 8 bit → Permite $2^8 = 256$ combinaciones distintas.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

TABLA ASCII 128 CARACTERES

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	2	3	4	5	6	7	8	9	LF 10	11	12	CR 13	14	15
0001	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0010	SP 32	!	"	#	S 35	% 37	& 38	*	()	*	+	-	.	/	47
0011	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	:	;	<	=	>	?
0100	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
0101	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	 91	\ 92	 93	^ 94	~ 95
0110	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
0111	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	 124	{ 125	~ 126	127

Caracteres 0-31: no imprimibles (p.e. 10: LF, fin de linea; 13:CR, "retorno de carro")

Caracter 32: "Espacio en blanco" (SP)

Caracter 127: no imprimible (DEL)

Bits de zona: Los primeros 4 bits de mayor significación en cada combinación se denominan **bits de zona** y en la tabla de doble entrada **numeran las filas 0 a 7**.

Bits de dígitos: Los últimos 4 bits (3, 2, 1 y 0) en cada combinación se denominan **bits de dígito** y en la **tabla numeran las columnas de 0 a 15, o de 0 a F en hexadecimal**.

Ejemplo:

$$\text{"A"} = 0100\ 0001 = 41_{(\text{h})} = 65_{(\text{d})}$$

TABLA ASCII 128 EXTENDIDO

Caracteres ASCII de control		Caracteres ASCII imprimibles		ASCII extendido (Página de código 437)	
00	NULL (carácter nulo)	32	espacio	128	ç
01	SOH (inicio encabezado)	33	!	129	ü
02	STX (inicio texto)	34	"	130	é
03	ETX (fin de texto)	35	#	131	â
04	EOT (fin transmisión)	36	\$	132	ã
05	ENQ (consulta)	37	%	133	à
06	ACK (reconocimiento)	38	&	134	â
07	BEL (timbre)	39	'	135	ç
08	BS (retroceso)	40	(136	ê
09	HT (tab horizontal)	41)	137	ë
10	LF (nueva línea)	42	*	138	è
11	VT (tab vertical)	43	+	139	ï
12	FF (nueva página)	44	,	140	î
13	CR (retorno de carro)	45	-	141	ì
14	SO (desplaza afuera)	46	.	142	Ã
15	SI (desplaza adentro)	47	/	143	À
16	DLE (esc.vinculo datos)	48	0	144	É
17	DC1 (control disp. 1)	49	1	145	æ
18	DC2 (control disp. 2)	50	2	146	Æ
19	DC3 (control disp. 3)	51	3	147	ô
20	DC4 (control disp. 4)	52	4	148	ö
21	NAK (conf. negativa)	53	5	149	ò
22	SYN (inactividad sinc)	54	6	150	ú
23	ETB (fin bloque trans)	55	7	151	û
24	CAN (cancelar)	56	8	152	ÿ
25	EM (fin del medio)	57	9	153	Ö
26	SUB (sustitución)	58	:	154	Ü
27	ESC (escape)	59	:	155	ø
28	FS (sep. archivos)	60	<	156	£
29	GS (sep. grupos)	61	=	157	Ø
30	RS (sep. registros)	62	>	158	×
31	US (sep. unidades)	63	?	159	f
127	DEL (suprimir)	95	-	191	ñ

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

CÓDIGOS DE REPRESENTACIÓN DECIMAL (BCD)

Los **códigos BCD** son convenciones que permiten la representación de números decimales (0 a 9) en bloques binarios de 4 bits. Existen varios códigos BCD

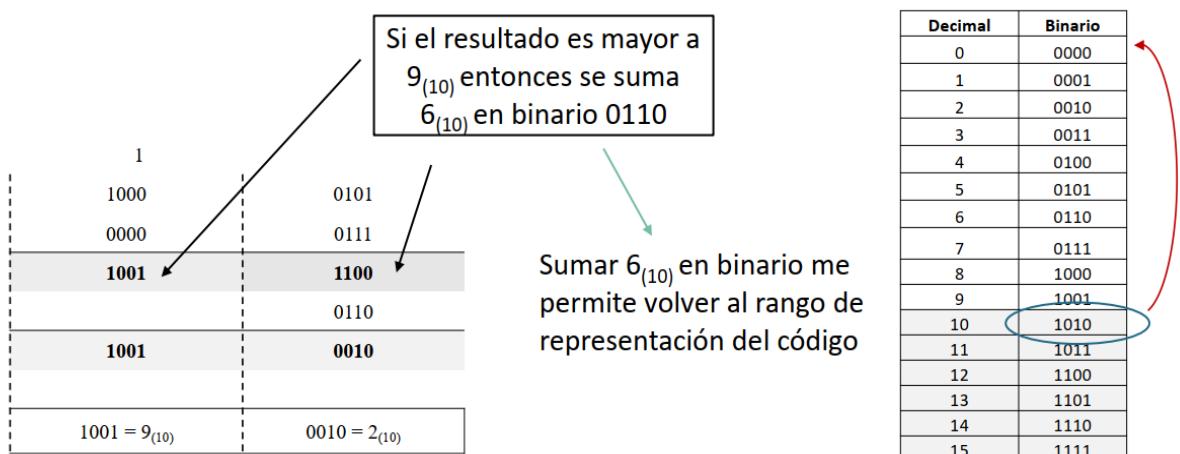
BCD PURO O NATURAL O BCD 8421

Valor decimal	BCD puro 8421	Valor decimal	BCD puro 8421
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Ejemplos	Valor decimal	Binario puro	BCD puro
	$15_{(10)}$	$1111_{(2)}$	$0001\ 0101_{(BCD)}$
	$256_{(10)}$	$2^8 = 100000000_{(2)}$	$0010\ 0101\ 0110_{(BCD)}$
	$1_{(10)}$	$1_{(2)}$	$0001_{(BCD)}$

SUMA EN BCD 8421

Ejemplo: $85_{(10)} + 7_{(10)} = 92_{(10)}$



UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

RESTA EN BCD 8421

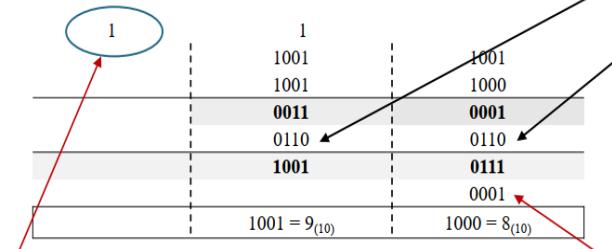
$$\text{Ejemplo: } 99_{(10)} - 1_{(10)} = 99_{(10)}$$

Se Complementa a 9 el valor negativo para realizar una suma $99_{(10)} + (-1_{(10)}) = 99_{(10)}$

Representación del -01

$$\begin{aligned} 0 &= 0000 + 0110 = 0110 \rightarrow 1001 \\ 1 &= 0001 + 0110 = 0111 \rightarrow 1000 \end{aligned}$$

Se suma 0110 y se invierten los bit para obtener C1



Si el resultado es mayor a $9_{(10)}$ entonces se suma $6_{(10)}$ en binario 0110

Sumar $6_{(10)}$ en binario me permite volver al rango de representación del código

Se desprecia por exceso de 10^n

Sumo 0001 por uso de C10

Decimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

OTROS CÓDIGOS DE REPRESENTACIÓN DECIMAL (BCD)

BCD exceso 3: Se obtiene a partir del BCD puro, sumando un 3 binario a cada cifra decimal.

Es un código simétrico o autocomplementario.

Facilita la operación de hallar el complemento de un número (complemento restringido decimal, complemento lógico o negación) sólo con invertir sus dígitos.

Valor decimal	BCD exceso tres
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Código autocomplementario

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OTROS CÓDIGOS DE REPRESENTACIÓN DECIMAL (BCD)

BCD AIKEN o 2421:

Valor decimal	BCD 2421
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

C digo autocomplementario

Ejemplos	Valor decimal	BCD exceso tres	BCD 2421
	$15_{(10)}$	$0100\ 1000_{(BCD\ EXC-3)}$	$0001\ 1011_{(BCD\ 2421)}$
	$256_{(10)}$	$0101\ 1000\ 1001_{(BCD\ EXC-3)}$	$0010\ 1011\ 1100_{(BCD\ 2421)}$
	$1_{(10)}$	$0100_{(BCD\ EXC-3)}$	$0001_{(BCD\ 2421)}$

CÓDIGO DE REPRESENTACIÓN BINARIA DE NÚMEROS ENTEROS

Coma fija o punto fijo sin signo para representar números enteros positivos: Se representan como número binario.

Como este formato sólo permite números sin signo (son todos positivos), para la representación de un número se utiliza la totalidad de bits del formato.

n bits de magnitud

Rango de representación:

$$(0, 2^n - 1)$$

n bits de formato

Ejemplo: si quiero representar en un formato de n = 8 el valor $25_{(10)}$

00011001

Rango de representación:
 $(0, 2^n - 1) = (0, 255)$

n = 8 bits de formato

Es decir, 256 combinaciones de 0 y 1 que me permiten almacenar valores entre 0 y 255 en decimal.

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Coma o punto fijo con signo (enteros): Para poder representar variables definidas como Enteras, lo primero que se debe definir es el **convenio** de cómo se va a representar el signo.

Para los **convenios** que vamos a ver, se adopta que el bit más significativo (MSB) se lo utilice para identificar el signo, siendo 0 para los positivos y 1 para los negativos.

Por lo tanto, si una variable está definida como **entera** todo número positivo comenzará de izquierda a derecha con 0 y todo número negativo comenzará con 1.

Convenio de Signo y Magnitud y Convenio de Complemento a 1:

1. Para **convenio de signo y magnitud** se representa el número como positivo y solo se le cambia el signo.

Ejemplo: representar el -15 con convenio de signo y magnitud en formato n = 6.

Representamos el +15 como 001111 y luego para el -15 cambiamos el signo quedando 101111.

2. Para **convenio de complemento a 1** se representa el número como positivo y se le aplica el complemento a 1 para obtener el número negativo.

Ejemplo: representar el -15 con convenio complemento a 1 en formato n = 6.

Representamos el +15 como 001111 y luego para el -15 complementamos a 1 quedando 110000

Observaciones:

En estos convenios vistos, el número cero tiene dos representaciones, una para el +0 y otra para el -0

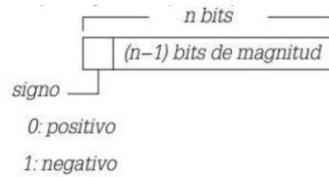
Signo y magnitud en n=8 queda 00000000 para el +0 y 10000000 para el -0

Complemento a 1 en n=8 queda 00000000 para el +0 y 11111111 para el -0

Es decir que se pierde una combinación para poder representar otro valor.

Como en las variables enteras sin signo, en las variables enteras también existirán los límites de representación según el formato adoptado y para ambos convenios, signo y magnitud y complemento a 1 será:

$$-(2^{n-1} - 1) \leq X \leq + (2^{n-1} - 1)$$



Ejemplo para n = 8 los límites serán desde -127 a +127. Fuera de este rango deberemos utilizar un formato mayor.

Convenio de Complemento a 2:

1. Se representa el número como positivo y para representar el negativo se le aplica el convenio de complemento a 2.

Ejemplo: representar el -15 con convenio de complemento a 2 en formato n = 6.

Representamos el +15 como 001111 y luego para el -15 complementamos a 2 a través del complemento a 1 + 1

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Convenio de Complemento a 2:

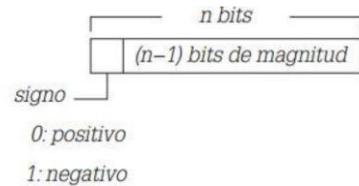
formato n = 6	
+15	001111 positivo
	110000 complementamos a 1
	1 sumamos 1
110001 Obtenemos el complemento a 2	

Observaciones:

En el convenio de complemento a dos a diferencia de los otros convenios, el número cero tiene solo una representación posible y es tomando por ejemplo formato n = 8 **00000000**.

Como en el caso de las variables enteras vistas, en las variables enteras bajo convenio de complemento a 2, también existirán los límites de representación según el formato adoptado.

$$-(2^{n-1}) \leq X \leq + (2^{n-1} - 1)$$



Ejemplo para n = 8 los límites serán desde - 128 a + 127. Vemos que hay un número más en los negativos que se puede representar en este convenio. Fuera de este rango deberemos utilizar un formato mayor.

Reglas generales del Convenio de Complemento a 2:

Si la variable definida como entera se puede representar el número positivo, siempre se va a poder representar el número negativo correspondiente.

Si la variable definida como entera no puede representar el número positivo, no se va a poder representar el número negativo correspondiente, salvo en complemento a dos.

BANDERAS LÓGICAS / FLAGS

Registro de estado (flags): Se trata de unos registros de memoria (en este caso ubicados en la ALU) en los que se deja constancia algunas condiciones que se dieron en la última **operación realizada** y que habrán de ser tenidas en cuenta en operaciones posteriores. Por ejemplo, en el caso de hacer una resta, tiene que quedar constancia si el resultado fue cero, positivo o negativo.

BANDERAS LÓGICAS / FLAGS

Definimos los siguientes flags:

Flag V (Overflow): indica si hubo o no overflow o desborde del registro, después de haberse llevado a cabo la operación. Si almacena un valor “1” significa que hubo overflow; si almacena un valor “0” significa que no hubo overflow.

En la suma o resta binaria:

Si los últimos 2 números de acarreo son **iguales** no hay overflow

Si los últimos 2 números de acarreo son **distintos**, hay overflow

Flag C (Carry): indica el acarreo o carry que se produce en la operación. Se pone en “1” si el ultimo bit de acarreo es igual a “1”, o se pone en “0” si el ultimo bit de acarreo es igual a “0”.

Flag S (Sign): indica el signo del resultado de la operación. Se pone en “1” si el resultado es negativo, o se pone en “0” si el resultado es positivo.

Flag Z (Zero): indica si el resultado de la operación es 0 o distinto de 0. Se pone en “1” si el resultado es 0, o se pone en “0” si el resultado es distinto de 0.

Flag P (Parity): si la cantidad de bits 1 del resultado de la operación es impar se pone un “1”, si la cantidad de bits 1 del resultado de la operación es par se pone un “0”.

Flag A (Auxiliary carry): indica el acarreo o carry que se produce del 4 bit. Se pone en “1” si C4 es igual a “1”, o se pone en “0” si el resultado es negativo.

DESBORDAMIENTO

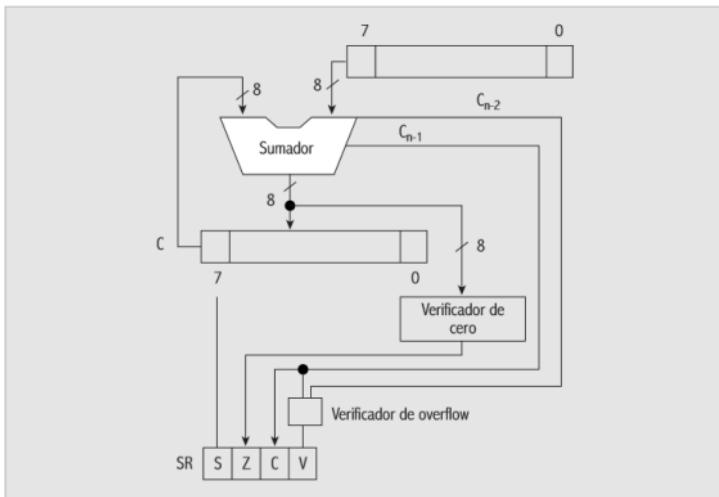
Overflow: Para las formas de variable vistas (Enteras Sin Signo (Naturales), Enteras con Signo y Reales, cuando el **resultado** de una **operación** supera los límites definidos por el rango (mayor que el máximo o menor que el mínimo), entonces **es incorrecto**.

Este error se conoce como **overflow** de resultado y actualiza un bit (flag de overflow) en un registro asociado a la ALU, conocido como registro de estado o status register.

Underflow: Solo para variables Reales, cuando el resultado de una operación cae en el hueco definido por los límites inferiores del rango (valores muy cercanos a cero, del que se excluye el cero). El error se conoce como **underflow** de resultado.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

ESQUEMA DE REGISTROS PARA SUMA DE 8 BITS



ALU: Unidad Aritmético lógica

$$A = -117_{(10)} \quad B = +68_{(10)}$$

$$(-117) + 68 = -49$$

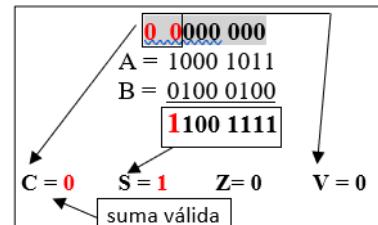


Fig. 4.1. Esquema de registros para suma de 8 bits.

Grafico del libro Arquitectura de Computadoras de Patricia Quiroga. Capítulo 4

OPERACIONES CON VARIABLES ENTERAS CON SIGNO (CONVENIO C2)

Ejemplo 1: Suma de 2 números enteros positivos:

Operación Suma paso a paso

R = A + B Donde A, B y R son variables enteras signadas
formato n = 8

$$A = 116 \text{ y } B = 5$$

$$116_{(10)} + 5_{(10)} = 121_{(10)}$$

1) Paso a binario los operandos.

$$\begin{array}{rcl} 116_{(10)} & = & 1110100_{(2)} \\ 5_{(10)} & = & 101_{(2)} \end{array}$$

2) Completo el formato a n=8 bits

$$\begin{array}{rcl} 116_{(10)} & = & 1110100_{(2)} = 01110100_{(2)} \\ 5_{(10)} & = & 101_{(2)} = 00000101_{(2)} \end{array}$$

3) C2 a los valores negativos

$$\begin{array}{rcl} 116_{(10)} & = & 01110100_{(2)} \\ 5_{(10)} & = & 00000101_{(2)} \end{array} \quad \boxed{\text{MP}}$$

Pregunta: La instrucción es Suma o Resta?
En este caso Suma

4) La ALU Realiza la suma en formato a n=8 bits
y entrega además los flags

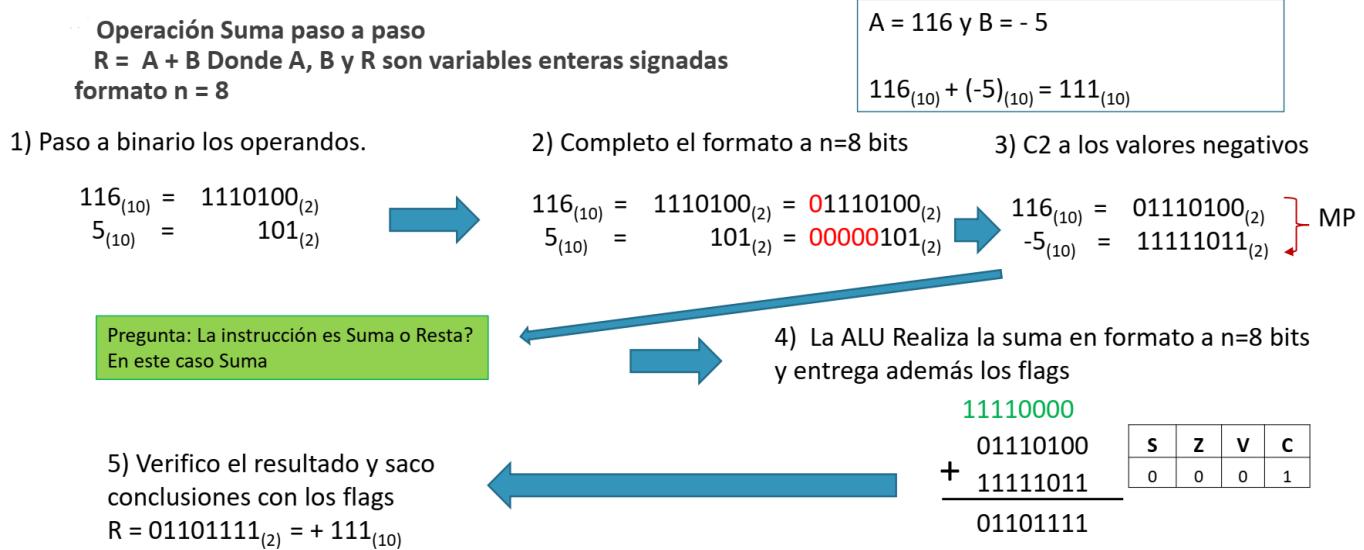
$$\begin{array}{r} 00000100 \\ 01110100 \\ + 00000101 \\ \hline 01111001 \end{array} \quad \begin{array}{|c|c|c|c|} \hline & S & Z & V & C \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

5) Verifico el resultado y saco conclusiones con los flags
 $R = 01111001_{(2)} = + 121_{(10)}$

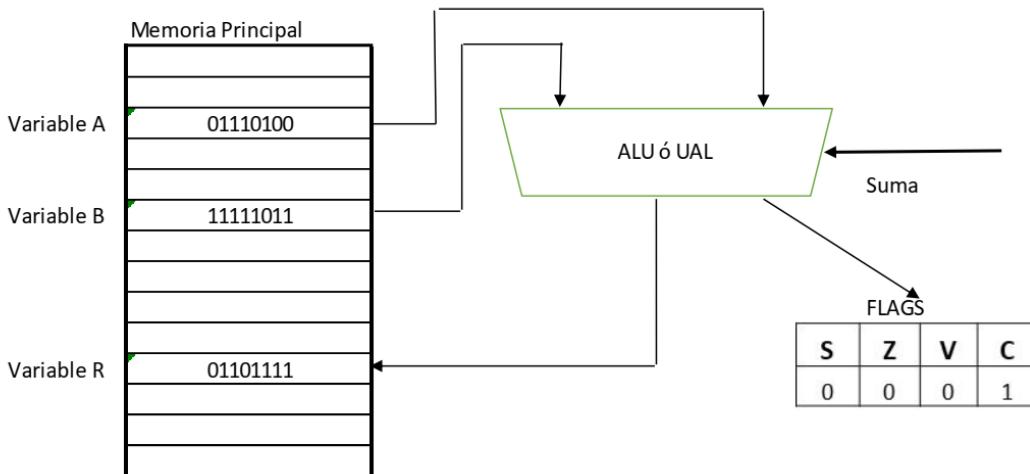
UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OPERACIONES CON VARIABLES ENTERAS CON SIGNO (CONVENIO C2)

Ejemplo 2: Suma de un número entero positivo con otro entero negativo:



Ejemplo 2 visto de forma gráfica en la ALU:



UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OPERACIONES CON VARIABLES ENTERAS CON SIGNO (CONVENIO C2)

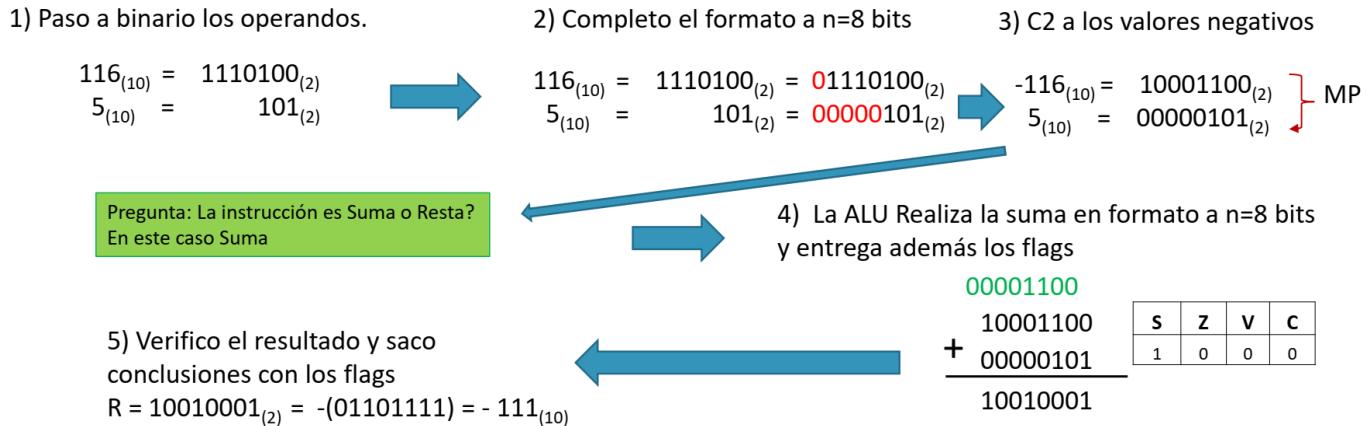
Ejemplo 3: Suma de un número entero negativo con otro entero positivo

Operación Suma paso a paso

R = A + B Donde A, B y R son variables enteras signadas
formato n = 8

$$A = -116 \text{ y } B = 5$$

$$(-116)_{(10)} + 5_{(10)} = -111_{(10)}$$



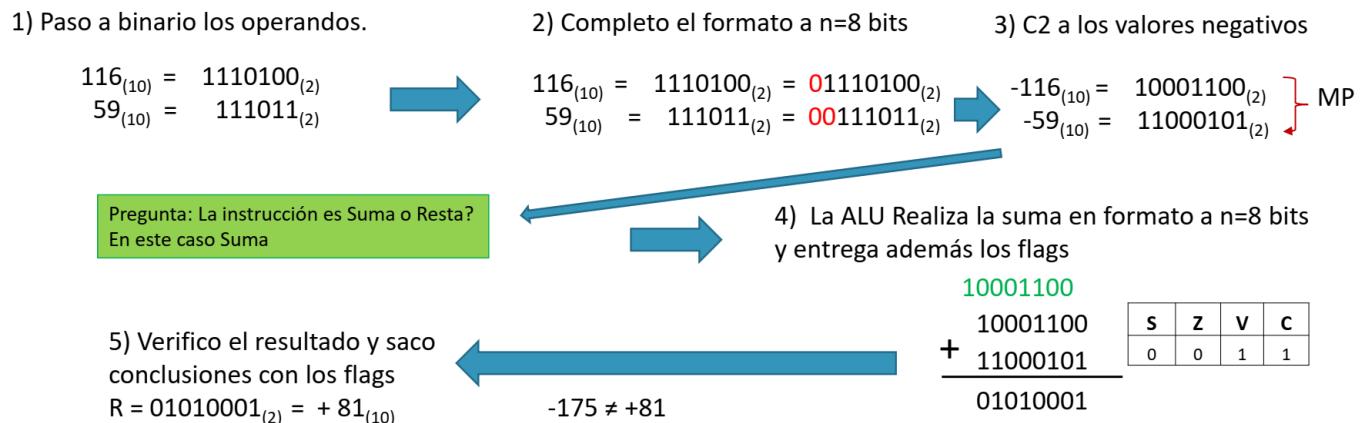
Ejemplo 4: Suma de 2 números enteros negativos

Operación Suma paso a paso

R = A + B Donde A, B y R son variables enteras signadas
formato n = 8

$$A = -116 \text{ y } B = -59$$

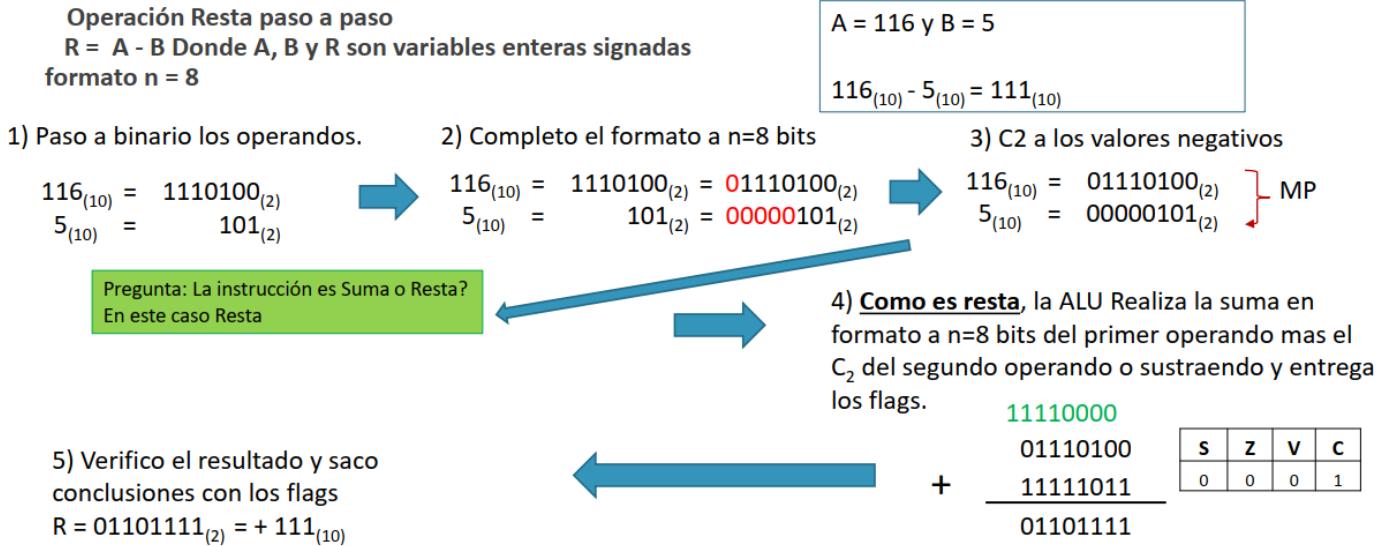
$$(-116)_{(10)} + (-59)_{(10)} = -175_{(10)}$$



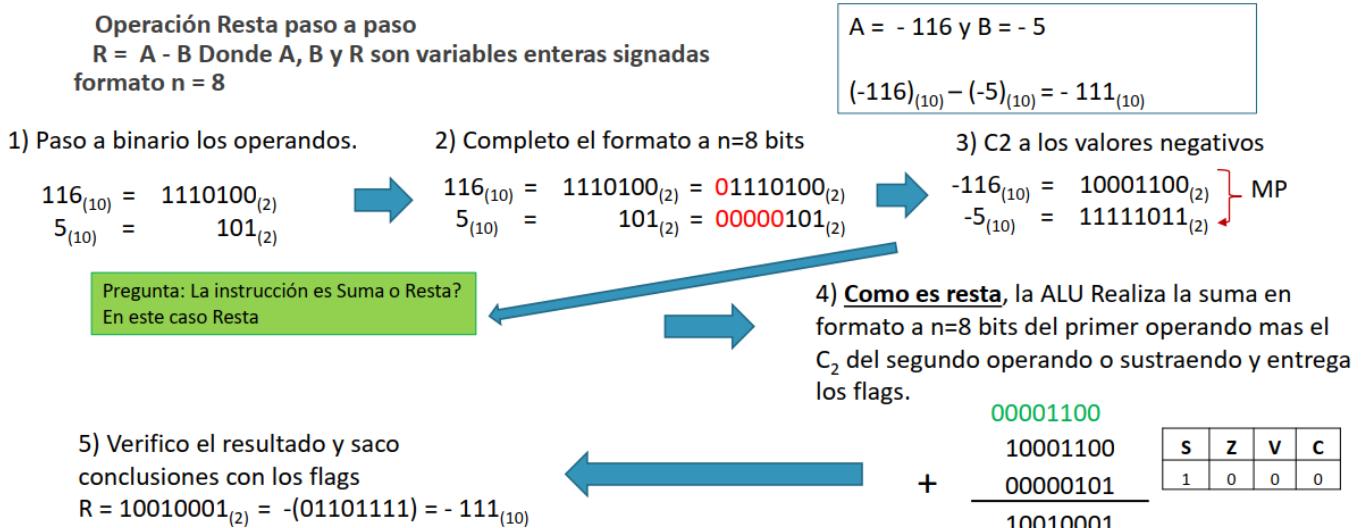
UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OPERACIONES EN CONVENIO PUNTO FIJO CON C2

Ejemplo 1: Resta de 2 números enteros positivos:



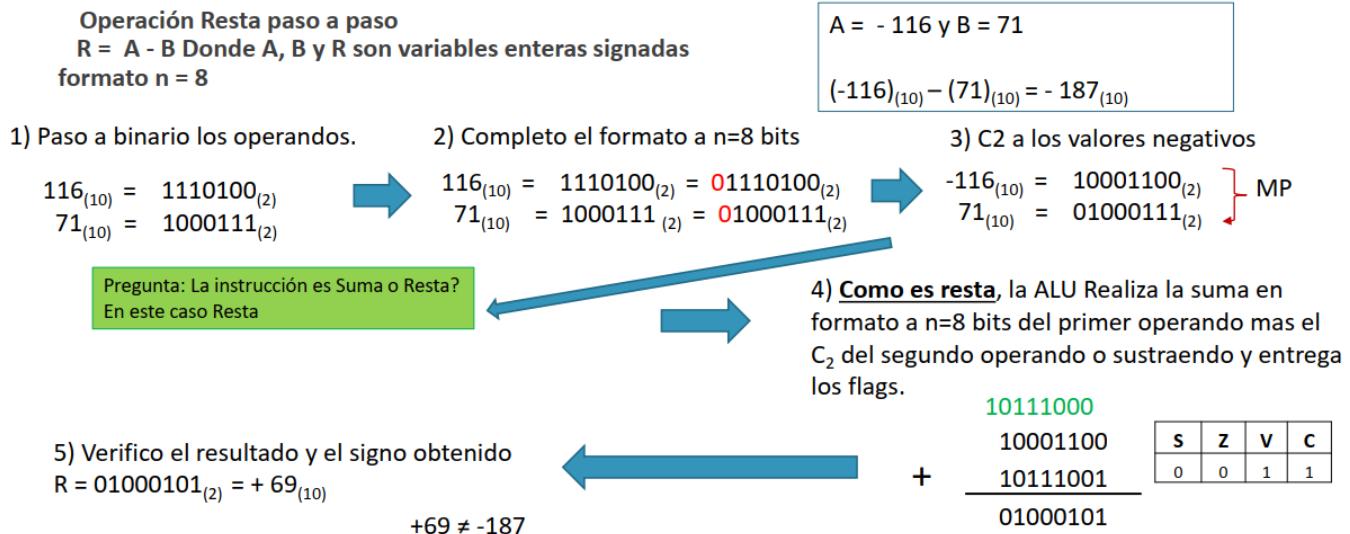
Ejemplo 2: Resta de 2 números enteros negativos:



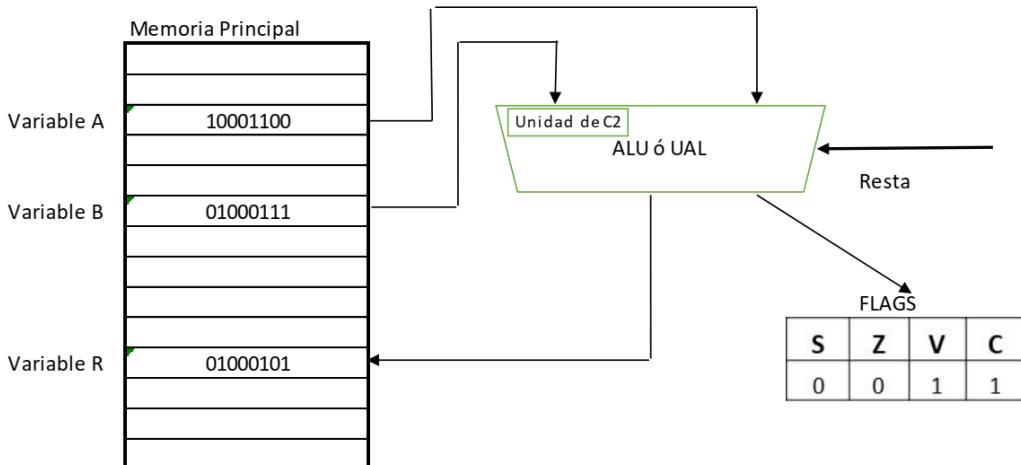
UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OPERACIONES EN CONVENIO PUNTO FIJO CON C2

Ejemplo 3: Resta de un número entero negativo con otro entero positivo:



Ejemplo 3 visto de forma gráfica en la ALU:



UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

ENDIANNES - EXTREMIDAD

Referencia al orden en el que se almacenan los bytes dentro de la memoria.

Si tenemos la siguiente cadena en hexadecimal 90AB12CD para almacenar en la memoria, según el orden:

Big Endian → Motorola

90AB12CD
----->

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	90	10010000
xxx1	AB	10101011
xxx2	12	00010010
xxx3	CD	11001101

Little Endian → Intel

90AB12CD
-----<

Representación de la Dirección o posición de memoria	Valor almacenado (hexadecimal)	Valor almacenado (binario)
xxx0	CD	11001101
xxx1	12	00010010
xxx2	AB	10101011
xxx3	90	10010000

Atención:
No se invierten los octetos o bytes.
Se almacena CD, no DC;
Se almacena 12, no 21;
y así sucesivamente...

Los dos → ARM y Power PC

OPERACIONES EN CONVENIO PUNTO FIJO ENTERAS SIN SIGNO (NATURALES)

Operación Resta paso a paso

R = A - B Donde A, B y R son variables enteras sin signo (Naturales)
formato n = 8

1) Paso a binario los operandos.

$$20_{(10)} = 10100_{(2)}$$

$$160_{(10)} = 10100000_{(2)}$$

2) Completo el formato a n=8 bits

$$20_{(10)} = 10100_{(2)} = 00010100_{(2)}$$

$$160_{(10)} = 10100000_{(2)} = 10100000_{(2)}$$

$$A = 20 \text{ y } B = 160$$

$$20_{(10)} - 160_{(10)} = -140_{(10)}$$

3) En MP como Ent. sin signo

$$20_{(10)} = 00010100_{(2)}$$

$$160_{(10)} = 10100000_{(2)}$$

Pregunta: La instrucción es Suma o Resta?
En este caso Resta

4) **Como es resta**, la ALU Realiza la suma en formato a n=8 bits del primer operando mas el C₂ del segundo operando o sustraendo y entrega los flags.

00000000	+	00010100	01100000	01110100

5) Verifico el resultado y saco conclusiones con los flags

$$R = 01110100_{(2)} = +116_{(10)}$$

$$+116 \neq -140$$

Observaciones:

Si durante la operación binaria, el **Flag de Carry = 1** y el **Flag de Signo = 0** entonces se le debe aplicar el complemento a 2 (C2) a ese resultado para que el resultado sea correcto.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

OPERACIONES EN CONVENIO PUNTO FIJO

En las **operaciones de Enteros con Signo** los **Flags** importantes son el **Z, S y V** donde:

Z: indica si es el resultado es cero o no ($Z=1$ resultado 0, $Z=0$ resultado distinto de cero)

S: indica si el resultado es Positivo o Negativo ($S=0$ positivo, $S=1$ negativo)

V: indica si el resultado de la operación de la instrucción excede los límites de representación para el formato dado.

En las **operaciones de Enteros sin Signo (Naturales)** los **Flags** importantes son en **Z y C**

Z: indica si es el resultado es cero o no ($Z=1$ resultado 0, $Z=0$ resultado distinto de cero)

C: indica si el resultado de la operación de la instrucción excede los límites de representación para el formato dado. (En este caso depende si es operación suma o resta). **Para el caso de resta** $C=1$ significa que el resultado está dentro de los límites de representación y $C=0$ significa que el resultado está fuera de los límites de representación. **Para el caso de suma**, será al revés.

COMPARACION DE VARIABLES

La **comparación de variables** se la usa para la **toma de decisiones** durante la **ejecución de un programa**, donde debe definir la secuencia de ejecución de la siguiente instrucción.

Determinar si dos variables A y B son iguales: Se realiza $A - B$ y se analiza el **flag Z**

a) Si $Z = 1$ entonces $A=B$

b) Si $Z = 0$ entonces $A \neq B$

Esto es tanto para variables enteras sin signo, como para las enteras con signo.

A partir del caso b) donde $A \neq B$, distinguimos cual variable es mayor (en este caso A respecto de B)

1) Análisis para las Enteras sin Signo: Se comprueba con el **flag C (Carry)**

a) Si $Z = 0$ y $C = 1$ entonces $A > B$

b) Si $Z = 0$ y $C = 0$ entonces $A < B$

2) Análisis para las Enteras con Signo: Se comprueba con el **Flag S (Signo)** y el **Flag V (Overflow)**

a) Si $Z = 0, V = 0$ y $S=0$ entonces $A > B$ Pero si $Z = 0, V = 0$ y $S=1$ entonces $A < B$.

b) Si $Z = 0$ y $V = 1$ y $S=0$ entonces $A < B$ Pero si $Z = 0, V = 1$ y $S=1$ entonces $A > B$.

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Variables Reales. Convenio de Punto Flotante.

Presenta una Semejanza con la notación científica o exponencial y sirve para representar números enteros y fraccionarios y números muy grandes y chicos.

Denominaciones:

Mantisa: La mantisa representa todos los bits del número sin coma o punto decimal.

Exponente: indica la posición del punto dentro del número

Signo: Puede ser positivo o negativo

Ejemplo: + 58,14 se puede a su vez representar como:

$58,14 \times 10^0$

$0,5814 \times 10^2$

$581,4 \times 10^{-1}$

Y así infinitas formas corriendo la coma y por ende cambiando el exponente.

La fórmula:

$$\pm M \cdot B^{\pm p}$$

Representa el criterio utilizado para todo **sistema posicional**, donde "M" es la mantisa, que podrá tratarse como **fracción pura** o como **entero**, según se **suponga** la coma a **izquierda** o a **derecha** de "M".

"B" es la base del sistema.

"P" es el exponente que indica la posición del punto en "M" y los signos "+" y "-" que la anteceden indican su desplazamiento a derecha o izquierda respecto del **origen**.

En la **memoria** sólo se almacenarán la mantisa "M" (parte fraccionaria como ya veremos) y la potencia "P" en binario (según el convenio).

La base "B" y el **origen** del punto siempre son determinados con anterioridad por el convenio y conocidos por los programas que utilizan esta representación.

La representación de datos en punto flotante **incrementa** el rango de números más allá de los límites físicos de los registros.

Por ejemplo, se sabe que en un registro de 16 bits el positivo mayor definido como entero signado que se puede representar es el +32767.

Con este convenio vamos a poder representar hasta un número positivo aproximadamente igual a 10^{+38} (1 seguido de 38 ceros)

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Variables Reales. Convenio de Punto Flotante.

Lo hacemos ahora para un **número binario**: 1101,101

$11,01101 \times 10^{10}$ (se aclara que a los fines prácticos y para facilidad de cálculos, el exponente lo pueden poner en decimal, sabiendo que es un número binario)

$11011,01 \times 10^{-1}$

$1,101101 \times 10^{11}$

Es decir, también infinitas combinaciones.

Variables Reales. Convenio de Punto Flotante (Convenio exceso 127)

Para la representación en este convenio de las **variables** definidas como **reales**:

1) Se define la representación normalizada: Adoptamos una de todas las posibles representaciones que podemos tener de un mismo número real.

Para este convenio y para el caso de los números binarios, la parte entera debe ser siempre 1 y el exponente será el valor positivo o negativo que resulte de desplazar la coma hacia izquierda o derecha para llegar a obtener la parte entera igual a 1.

Por lo tanto de todas las posibles combinaciones del número 1101,101 vamos a mover la coma (en este caso a la izquierda) quedando como expresión normalizada **$1,101101 \times 10^{11}$**

2) Representación en la Memoria principal:

Lo primero es saber que tiene su propio formato y el mismo es de 32 bits o cuatro octetos en lo que se llama **simple precisión**. El formato es el siguiente:

1 bit 8 bits $m = 23$ bits mantisa

S	$\pm p + 127$	
-----	---------------	--

1, (posición supuesta del bit implícito)

Para eliminar el signo del exponente se utilizan convenios que agregan al exponente un **exceso** igual a $2^{m-1}-1$.

Donde m es la cantidad de bits que posee el campo exponente (Característica)

Esta entidad nueva se denomina característica y es igual a:

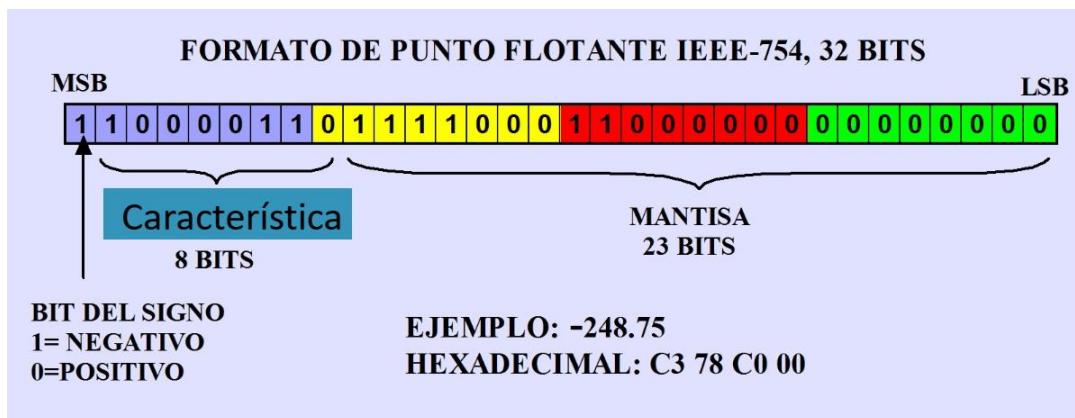
Característica = P + exceso

Siendo p el exponente de la expresión normalizada y el exceso dependiendo la precisión del convenio de punto flotante utilizado.

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Variables Reales. Convenio de Punto Flotante (Convenio exceso 127 – Estándar IEEE 754)

El siguiente caso se muestra que para simple precisión, al ser el campo del exponente de 8 bits el exceso a sumar al exponente p de la expresión normalizada será $2^{8-1}-1 = 127$. De acá sale el nombre de exceso 127.



Ejemplo: Representamos en memoria la variable A definida como Real en Simple precisión:

$$A = -53,25$$

1) Primero debemos pasar el decimal a Binario natural. **Tener en cuenta que punto flotante no trabaja con complemento.**

Da como resultado en binario A = - 110101,01

Luego normalizamos: $-1,1010101 \times 10^{101}$ donde el exponente p está también en binario que es el 5 decimal.

Con esto pasamos a la representación:

Campo Bit de Signo: $S = 1$ por ser número negativo

Campo exponente (Característica) $C = p + 127$, es decir $5 + 127 = 132$ que se representará con los 8 bits de dicho campo. (10000100)

Campo mantisa (es lo que está en la parte fraccionaria de la expresión normalizada) es 1010101

Con esto podemos ver cómo queda la representación de la variable en los 32 bits que utiliza este formato.

The diagram illustrates the binary representation of a floating-point number. It consists of three main parts: a green arrow pointing up labeled "Signo" pointing to the first bit; a red arrow pointing up labeled "Exponente excedida ó Característica" pointing to the next 8 bits; and a yellow arrow pointing up labeled "Mantisa (Se completa con ceros hasta completar los 23 bits)" pointing to the remaining 23 bits.

1 10000100 1010101000000000000000000

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Variables Reales. Convenio de Punto Flotante (Convenio exceso 127 – Estándar IEEE 754)

Valores que representan el cero y el infinito:

La representación del Número cero en este convenio es con los 32 bits iguales a cero

El infinito se representa con todos 1 en el campo exponente (característica) y todos 0 en la mantisa.

Los valores **mínimo y máximo** que puede tomar la característica y el exponente son:

Característica mínima 00000000 por lo tanto exponente mínimo -127

Característica máxima 11111111 por lo tanto el exponente máximo 128

Como los valores extremos de las características se los utiliza para representar el cero y el infinito, la misma para el resto de los valores puede variar entre 00000001 y 11111110 y esto da como resultado no poder representar números muy cercanos a cero.

Límites de representación:

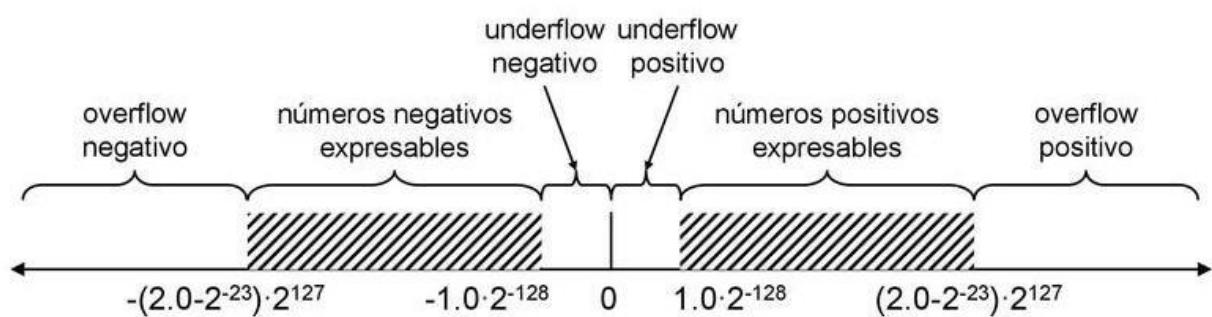
Como vimos en las variables Enteras sin signo (Naturales) y las Enteras, las variables reales tienen límites de representación.

Para simple precisión serán:

Mínimo aproximadamente -10^{38} y Máximo aproximadamente 10^{38}

Pero a raíz de los valores reservados para la Característica mencionados anteriormente resulta que números muy cercanos al cero no podrán representarse y los mismos son:

Entre -10^{-38} y 10^{-38} aproximadamente excluyendo el cero que si se puede representar.



UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Variables Reales. Convenio de Punto Flotante (Otros Convenios – IEEE 754)

Punto flotante a su vez puede trabajar con lo que se llama **Doble Precisión** y también con **Precisión Extendida**.

Para estos casos varía tanto el campo característica como el campo mantisa. En el primer caso se logra ampliar los límites de representación (poder representar números aún más grandes y más chicos), y en el segundo se aumenta la precisión con la cual vamos la variable se define (cuantos decimales de precisión va a tener)

	Cantidad de Bits				Exceso a sumar al exponente
	Signo	Característica	Mantisa	Total de bits	
SIMPLE PRECISION	1	8	23	32	127
DOBLE PRECISION	1	11	52	64	1023
PRECISION EXTENDIDA	1	15	64	80	16383

REPRESENTACIONES REDUNDANTES

El cambio de un bit en el almacenamiento o la manipulación de datos origina resultados erróneos.

Para detectar e incluso corregir estas alteraciones se usan códigos que agregan “**bits redundantes**”.

Por ejemplo, los bits de paridad se agregan al dato en el momento de su envío y son corroborados en el momento de su recepción por algún componente de la computadora, y lo mismo sucede al revés.

El resultado de “medir” la cantidad de “ruido” que puede afectar la información se representa con un coeficiente conocido como **tasa de error**.

De acuerdo con el valor de la tasa de error, se selecciona un código, entre los distintos desarrollados, para descubrir e incluso corregir los errores.

Estos códigos reciben el nombre de **códigos de paridad**.

Paridad vertical simple o a nivel carácter

Al final de cada carácter se incluye un bit, de manera que la suma de “unos” del carácter completo sea par (paridad par) o impar (paridad impar). Este tipo de codificación se denomina paridad vertical. Suele acompañar la transmisión de octetos en los periféricos y la memoria.

Los ejemplos muestran el bit de paridad que se agrega al octeto, en el caso de usar paridad par o impar:

00110010 | 1 se agrega un uno que permite obtener paridad par
01001011 | 0 se agrega un cero para lograr paridad par

Este código de detección de errores sólo se utiliza si la tasa de error en la cadena de bits que se han de transmitir no es superior a uno; esto es, si hay dos bits erróneos no permite detectarlos.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

CÓDIGOS DE REPRESENTACIÓN NO DECIMAL

Paridad entrelazada:

Utilizando en conjunto el código de paridad vertical con el horizontal se construye el código de paridad entrelazada, que además de detectar errores permite corregirlos.

Ejemplo: Supongamos una transmisión de cuatro caracteres en código ASCII de 7 bits, con paridad par.

El primer grupo indica cómo deberían llegar los caracteres a su lugar de destino, de no haberse producido errores en la transmisión.

El segundo grupo indica con un recuadro el bit que tuvo un error en la transmisión.

PH	PH
0 1 0 1 0	0 1 0 1 0
0 1 1 1 1	0 [0] 1 1 1 ——
1 1 0 1 1	1 1 0 1 1
1 0 1 1 1	1 0 1 1 1
1 1 1 0 1	1 1 1 0 1
1 0 1 1 1	1 0 1 1 1
0 1 1 0 0	0 1 1 0 0
<i>PV</i> – 0 1 1 1 1	<i>PV</i> – 0 1 1 1 1
<i>sin error</i>	<i>con error</i>

La forma de detectar el error es verificar tanto los bits del mensaje como los de paridad, generando un nuevo bit de paridad (en este caso llamado Verificador) Donde hay 1 es que hay error.

La forma de corregir el error es invertir el bit señalado como erróneo.

Si hay más de un error en la misma fila o columna, quizás se pueda detectar, pero no se podrá determinar con exactitud cuál es el error.

PH	PH	Verificador
0 1 0 1 0	0 1 0 1 0	0
0 1 1 1 1	0 [0] 1 1 1 ——	1
1 1 0 1 1	1 1 0 1 1	0
1 0 1 1 1	1 0 1 1 1	0
1 1 1 0 1	1 1 1 0 1	0
1 0 1 1 1	1 0 1 1 1	0
0 1 1 0 0	0 1 1 0 0	0
<i>PV</i> – 0 1 1 1 1	<i>PV</i> – 0 1 1 1 1	0
<i>sin error</i>	<i>con error</i>	

Verificador 0 1 0 0 0

BCD EMPAQUETADO Y BCD DESEMPAQUETADO

BCD Desempaquetado es utilizado en la entrada de datos desde periféricos: Se lo codifica como alfanumérico utilizando 1 byte para cada número. Como sobran 4 bits para cada número dentro del byte, que es el nible de mayor peso, se lo completa con la combinación 0011, es decir cada byte será por 0011xxxx donde xxx será el BCD Natural del número a representar.

Si observan lo indicado, coincide con la representación en ASCII de 8 bits para cada número.

UNIDAD 3 Y 4: REPRESENTACIÓN DE DATOS EN LA COMPUTADORA

BCD EMPAQUETADO Y BCD DESEMPAQUETADO

BCD Desempaquetado

Por ejemplo el 579 en BCD desempaquetado será:

00110101 00110111 00111001 es decir todos empiezan con 0011 y siguen con el BCD del número a representar.

Finalmente al inicio de la representación se agregará el signo + ó – también codificado según tabal ASCII para dichos caracteres.

Ejemplo: -2584 en BCD desempaquetado será 00101101 00110010 00110101 00111000 00110100 expresado en Hexadecimal 2D 32 35 38 34

+653 en BCD desempaquetado será 00101011 00110110 00110101 00110011 expresado en Hexadecimal 2B 36 35 33

BCD Empaquetado es una forma de aprovechar mejor el formato que el desempaquetado. Se codifican 2 dígitos decimales por byte. En este caso el último medio byte será el del signo y no al principio como en el desempaquetado.

Para el signo se utiliza 1011 (B) para los positivos y 1101 (D) para los negativos.

Los datos empaquetados deben emplear un número entero de bytes, por lo tanto de ser necesario para completar en cantidad entera de bytes, se deben agregar 4 ceros a la izquierda del digito BCD más significativo.

Si tomamos los ejemplos anteriormente para BCD desempaquetado, en Empaquetado van a quedar de la siguiente manera:

Ejemplo: -2584 en BCD Empaquetado será 00000010 01011000 01001101 ver que completo el primer byte con ceros y el signo – está al final. Expresado en Hexadecimal 02 58 4D

+653 en BCD Empaquetado será 01100101 00111011 En este caso no hizo falta rellenar con 4 ceros al principio. Expresado en Hexadecimal 65 3B

UNIDAD 5: ÁLGEBRA DE BOOLE

INTRODUCCIÓN

Los circuitos en computadoras digitales y otros sistemas digitales se diseñan y se analizan a través del álgebra de Boole.

Análisis: Es una forma concisa de describir el funcionamiento de los circuitos digitales.

Diseño: Dada una función deseada, se puede aplicar el **álgebra de Boole** para desarrollar una función simplificada de la dada e implementarla a través de circuitos digitales.

UNIDAD 5: ÁLGEBRA DE BOOLE

DEFINICIONES

Variable de Boole: Aquella variable que solo puede tomar dos estados.

Ejemplos:

Si / No

Verdadero / Falso

Encendido / Apagado

0 / 1

Función de Boole: Es aquella que está formada por **variables de Boole** y cuyo valor de la función, al ser de Boole, también podrá tomar dos estados posibles (Binario).

Operadores booleanos o lógicos:

SUMA LOGICA (+; OR)

PRODUCTO LOGICO (\bullet ; AND)

COMPLEMENTO O NOT (-)

Estos operadores y cualquier función booleana quedan definidos mediante sus **Tablas de Verdad**.

Tabla de Verdad: Es la representación a través de una tabla, donde se muestran los valores de la función para todas las posibles combinaciones de las variables que conforman la misma (también se dice combinaciones de entrada)

También se usan **expresiones literales** (polinómica y factorial) y expresiones simbólicas.

La **forma canónica** de una función booleana es una expresión cuyos términos contienen la totalidad de las variables del problema.

Se dice que **dos funciones** son iguales cuando tienen la misma tabla de verdad.

Funciones booleanas: Conjunto de variables booleanas vinculadas entre sí mediante los operadores lógicos

Compuertas Lógicas: dispositivos físicos que implementan una función booleana simple.

PROPIEDADES DEL ÁLGEBRA DE BOOLE

Leyes conmutativas

Ambas operaciones binarias (suma y producto) son conmutativas, esto es que si a y b son elementos del Álgebra se verifica que:

$$a + b = b + a$$

$$a \cdot b = b \cdot a$$

Leyes Distributivas

Cada operación binaria (suma y producto) es distributiva respecto de la otra:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

Leyes de Identidad

Dentro del Álgebra existen dos elementos neutros, el 0 y el 1, que cumplen la propiedad de identidad con respecto a cada una de las operaciones binarias:

$$a + 0 = a$$

$$a \cdot 1 = a$$

UNIDAD 5: ÁLGEBRA DE BOOLE

PROPIEDADES DEL ÁLGEBRA DE BOOLE

❖ Propiedad o Ley de Complementación o Inverso

Para cada elemento a del Álgebra, existe un elemento denominado a negada, tal que:

$$a + \bar{a} = 1$$

a y \bar{a} no pueden ser cero al mismo tiempo,

$$a \cdot \bar{a} = 0$$

a y \bar{a} no pueden ser uno al mismo tiempo. Estas dos leyes definen el complemento de una variable.

❖ Propiedad o Ley de ídem potencia

Para cada elemento a de un Álgebra de Boole se verifica que:

$$a + a = a$$

$$a \cdot a = a$$

❖ Propiedad o Ley Anulación de la variable

Para cada elemento del Álgebra de Boole se verifica que:

$$a + 1 = 1$$

$$a \cdot 0 = 0$$

Según esta propiedad y las “leyes de identidad” citadas anteriormente, se deduce que:

$$0 + 0 = 0 \quad 0 \cdot 0 = 0$$

$$0 + 1 = 1 \quad 0 \cdot 1 = 0$$

$$1 + 1 = 1 \quad 1 \cdot 1 = 1$$

❖ Propiedad o Ley de absorción

Para cada par de elementos del Álgebra de Boole, a y b , se verifica que:

$$a + (a \cdot b) = a$$

$$a \cdot (a + b) = a$$

Confeccionando la tabla de verdad se puede demostrar, por ejemplo, para la primera igualdad:

Tabla 5-5			
a	b	$a \cdot b$	$a + a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Teorema 4 - Ley de absorción

Para cada par de elementos del Álgebra de Boole, a y b , se verifica que:

$$a + (a \cdot b) = a$$

$$a \cdot (a + b) = a$$

Confeccionando la tabla de verdad se puede demostrar, por ejemplo, para la primera igualdad (tabla 5-5):

Tabla 5-5			
a	b	$a \cdot b$	$a + a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Teorema 5 – Propiedad Asociativa

En el Álgebra de Boole la suma y el producto son asociativos:

$$a + b + c = (a + b) + c = a + (b + c)$$

$$a \cdot b \cdot c = (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

UNIDAD 5: ÁLGEBRA DE BOOLE

PROPIEDADES DEL ÁLGEBRA DE BOOLE

❖ Propiedad Asociativa

En el Álgebra de Boole la suma y el producto son asociativos:

$$a + b + c = (a + b) + c = a + (b + c)$$

$$a \cdot b \cdot c = (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

❖ Propiedad de la doble negación o Ley de involución

Para todo elemento a de un Álgebra de Boole, se verifica que:

$$\bar{\bar{a}} = a$$

a	\bar{a}	$\bar{\bar{a}}$
0	1	0
1	0	1

Teorema o Ley de DeMorgan

En todo Álgebra de Boole se verifican las siguientes igualdades permiten transformar sumas en productos y productos en sumas:

$$1) \overline{a+b} = \bar{a} \cdot \bar{b}$$

$$2) \overline{a \cdot b} = \bar{a} + \bar{b}$$

$$\overline{a+b+c+\dots} = \bar{a} \cdot \bar{b} \cdot \bar{c} \cdot \dots$$

$$\overline{a \cdot b \cdot c \cdot \dots} = \bar{a} + \bar{b} + \bar{c} + \dots$$

Consecuencia:

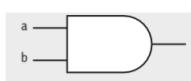
$$1) \overline{\overline{a+b}} = \overline{\overline{\bar{a} \cdot \bar{b}}} = a+b$$

$$2) \overline{\overline{a \cdot b}} = \overline{\overline{\bar{a} + \bar{b}}} = a \cdot b$$

COMPUERTAS LÓGICAS O BOOLEANAS

AND

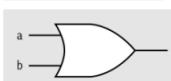
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



$$f(a, b) = a \cdot b$$

OR

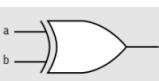
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



$$f(a, b) = a + b$$

XOR

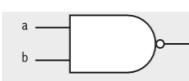
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	0



$$f(a, b) = a \oplus b$$

NAND

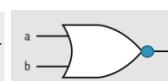
A	B	A.B
0	0	1
0	1	1
1	0	1
1	1	0



$$f(a, b) = \bar{a} \cdot \bar{b}$$

NOR

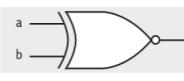
A	B	A+B
0	0	1
0	1	0
1	0	0
1	1	0



$$f(a, b) = \bar{a} + \bar{b}$$

XNOR

A	B	A+B
0	0	1
0	1	0
1	0	0
1	1	1



$$f(a, b) = \bar{a} \oplus \bar{b}$$

UNIDAD 5: ÁLGEBRA DE BOOLE

MINITÉRMINOS Y MAXITÉRMINOS

Minitérmino se define a un **producto** donde aparecen **todas las variables** que intervienen en la función afectadas o no cada una por la negación.

Supongamos una función $F(a,b,c,d)$

Cuales de los siguientes son minitérminos:

\bar{a}	b	c	d	<input checked="" type="checkbox"/>
\bar{a}	c	\bar{d}		<input type="checkbox"/>
\bar{a}	b	c	d	<input type="checkbox"/>
\bar{a}	b	$(c + d)$		<input type="checkbox"/>
\bar{a}	\bar{b}	\bar{c}	\bar{d}	<input checked="" type="checkbox"/>

Maxitérmino se define a una **suma** donde aparecen **todas las variables** que intervienen en la función afectadas o no cada una por la negación.

Supongamos una función $F(a,b,c,d)$

Cuales de los siguientes son maxitérminos:

$a + b + c$	<input type="checkbox"/>
$\bar{a} + b + \bar{c} + d$	<input checked="" type="checkbox"/>
$\bar{a} + \bar{b} + c + d$	<input type="checkbox"/>
$\bar{a} + \bar{b} + \bar{c} + d$	<input checked="" type="checkbox"/>
$\bar{a} + b + \bar{c} + \bar{d}$	<input type="checkbox"/>

FORMAS NORMALES O CANÓNICAS DE UNA FUNCIÓN

La aplicación principal de los minitérminos es poder representar una función como la suma lógica de aquellos minitérminos que en su tabla de verdad, la función tome el valor 1. Esto se llama expresar una función en su forma normal disyuntiva (FND)".

Vamos a asignar un **maxitérmino o suma canónica** a cada una de las combinaciones de la tabla de verdad por medio de la suma lógica, tal que, dicha suma para las combinaciones de las variables de entrada sea 0.

<i>a</i>	<i>b</i>	<i>Maxitérminos</i>	
0	0	$a + b$	M0
0	1	$a + \bar{b}$	M1
1	0	$\bar{a} + b$	M2
1	1	$\bar{a} + \bar{b}$	M3

UNIDAD 5: ÁLGEBRA DE BOOLE

TABLAS DE VERDAD

Una tabla de verdad es: "Una lista ordenada de las 2^n combinaciones distintas de ceros y unos, que se pueden obtener de la combinación del valor de n variables binarias". Para 3 variables la cantidad de combinaciones binarias distintas es de $2^3 = 8$

A	B	C	Z
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

Ejemplo de aplicación de minitérminos: Queremos diseñar un circuito que tiene 3 entradas y una salida cuyo valor es igual a 1 si por lo menos dos entradas adyacentes son iguales a 1.



1ro) Planteamos la tabla de verdad:

A	B	C	S
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2do) Identificamos los minitérminos para los cuales la función S vale 1

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$m_3 = \bar{A} \bar{B} C$

$m_6 = A \bar{B} \bar{C}$

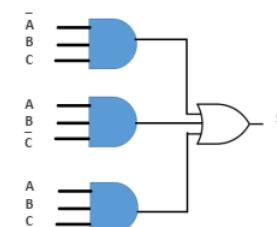
$m_7 = A B C$

3ro) Se realiza la suma de los minitérminos cuya función vale 1

$$S = \bar{A} B C + A \bar{B} \bar{C} + A B C$$

Esta es la expresión de la función en su forma normal disyuntiva FND (suma de productos)

4to) Se implementa el circuito (Compuertas AND y una OR)



UNIDAD 5: ÁLGEBRA DE BOOLE

FORMAS NORMALES O CANÓNICAS DE UNA FUNCIÓN

Forma normal disyuntiva:

“Si se expresa una función como la suma lógica de aquellos minitérminos que en su tabla de verdad tengan valor 1, se obtiene la expresión de su forma normal disyuntiva FND”.

Suma de productos que se implementan a través de compuertas AND (productos) que confluyen en una compuerta OR (realiza la suma de dichos productos)

Forma normal conjuntiva

“Si se expresa una función como el producto lógico de aquellos maxitérminos que en su tabla de verdad tengan valor 0, se obtiene la expresión de su forma normal conjuntiva (FNC)”.

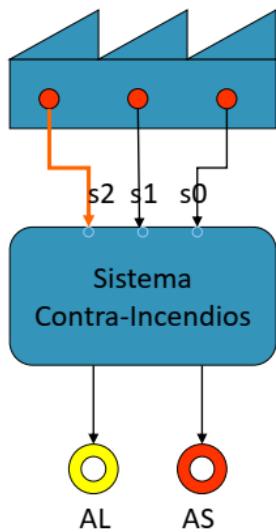
Producto de sumas que se implementan a través de compuertas OR (sumas) que confluyen en una compuerta AND (realiza el producto de dichas sumas)

La FND y la FNC se denominan formas normales o canónicas de una función.

EJEMPLO DE FUNCIONES BOOLEANAS

Ejemplo 1: El sistema de seguridad contra incendios de un depósito funciona en base a tres sensores S0, S1 y S2. Cuando dos de estos sensores están activados (en “1”) se enciende una alarma luminosa [AL]. Además, si S2 se activa, también se enciende la alarma sonora [AS].

Tabla de Verdad



s2	s1	s0	AL	AS
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

UNIDAD 5: ÁLGEBRA DE BOOLE

EJEMPLO DE FUNCIONES BOOLEANAS

Ejemplo 1: Formas Literales:

Expresión Literal Completa –Suma de Productos

$$AL = \overline{S_2} \cdot S_1 \cdot S_0 + S_2 \cdot \overline{S_1} \cdot S_0 + S_2 \cdot S_1 \cdot \overline{S_0} + S_2 \cdot S_1 \cdot S_0$$

$$AS = S_2 \cdot \overline{S_1} \cdot \overline{S_0} + S_2 \cdot \overline{S_1} \cdot S_0 + S_2 \cdot S_1 \cdot \overline{S_0} + S_2 \cdot S_1 \cdot S_0$$

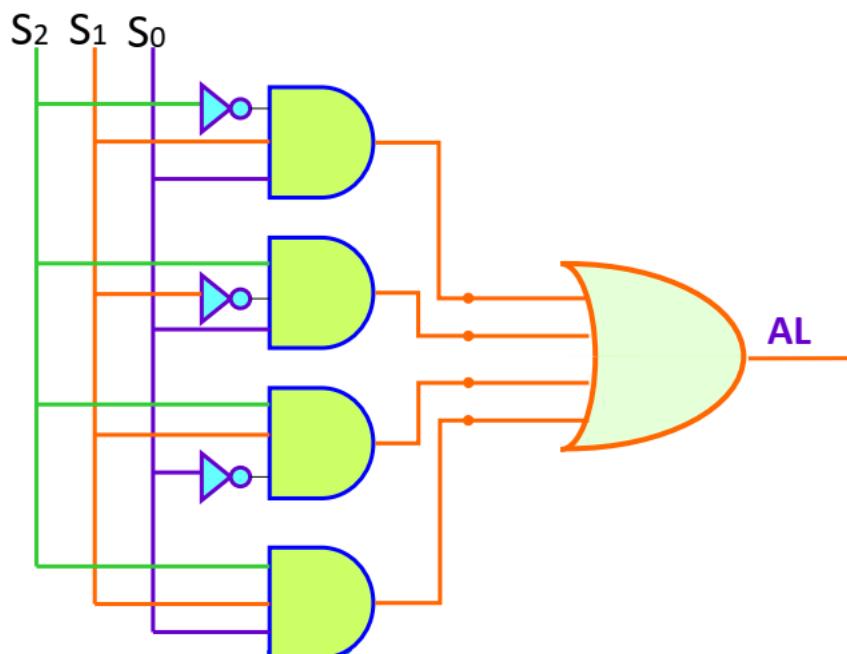
Expresión Literal en Minitérminos

$$AL = m_3 + m_5 + m_6 + m_7$$

$$AS = m_4 + m_5 + m_6 + m_7$$

Dado que todos los términos de esta forma POLINÓMICA contienen todas las variables, esta expresión se denomina CANÓNICA

Ejemplo 1: Circuito Lógico:

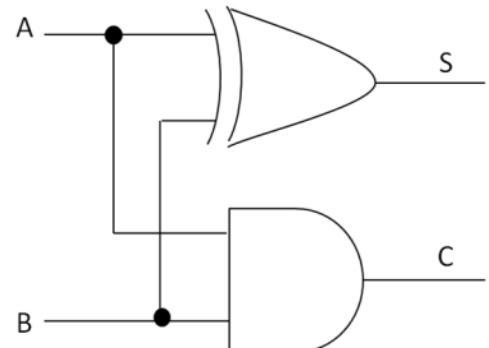
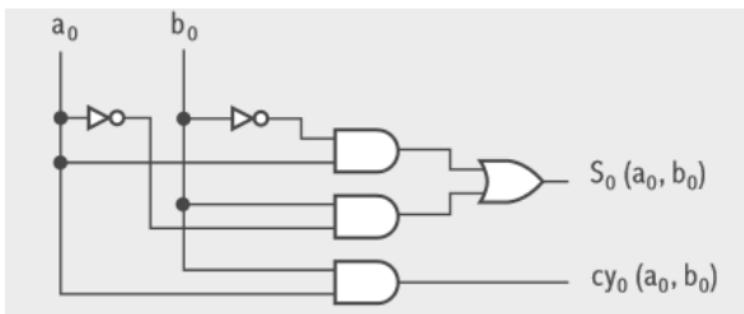
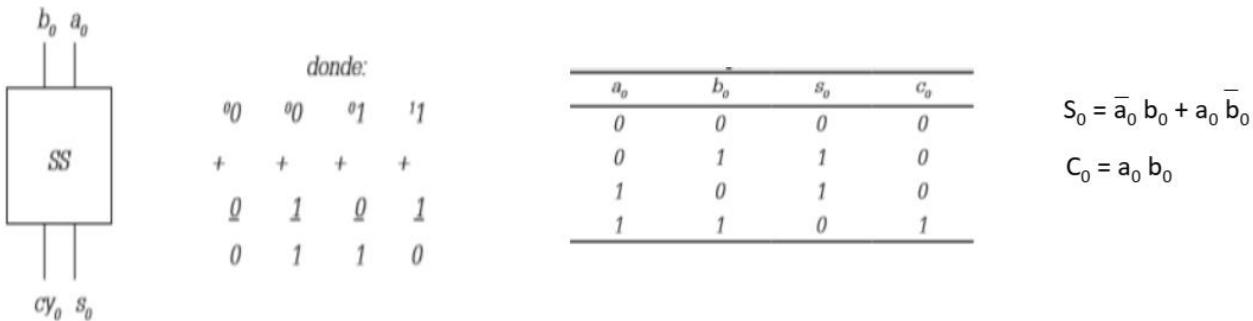


UNIDAD 5: ÁLGEBRA DE BOOLE

CIRCUITO SEMISUMADOR

Circuito semisumador (SS) o Half Adder (HA)

Es un circuito cuya función es sumar 2 bits sin tener en cuenta el acarreo anterior. Tiene dos salidas, una representa la suma y la otra, el valor del acarreo (carry).



CIRCUITO SUMADOR COMPLETO

Circuito sumador completo (SC) o Full Adder (FA)

Este circuito tiene como finalidad sumar 2 bits (independientemente de la columna que quiera sumar dentro de un número binario de n bits) y el acarreo anterior, generando como salidas el resultado de la suma y el nuevo acarreo.

Genéricamente:

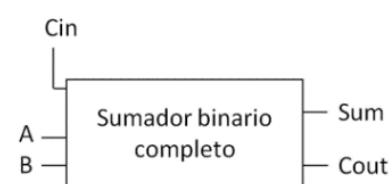
$$A = A_n-1 \dots A_1 A_0$$

+

$$B = B_n-1 \dots B_1 B_0$$

$$S = S_n-1 \dots S_1 S_0$$

Cin	A	B	S	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



UNIDAD 5: ÁLGEBRA DE BOOLE

CIRCUITO SUMADOR COMPLETO

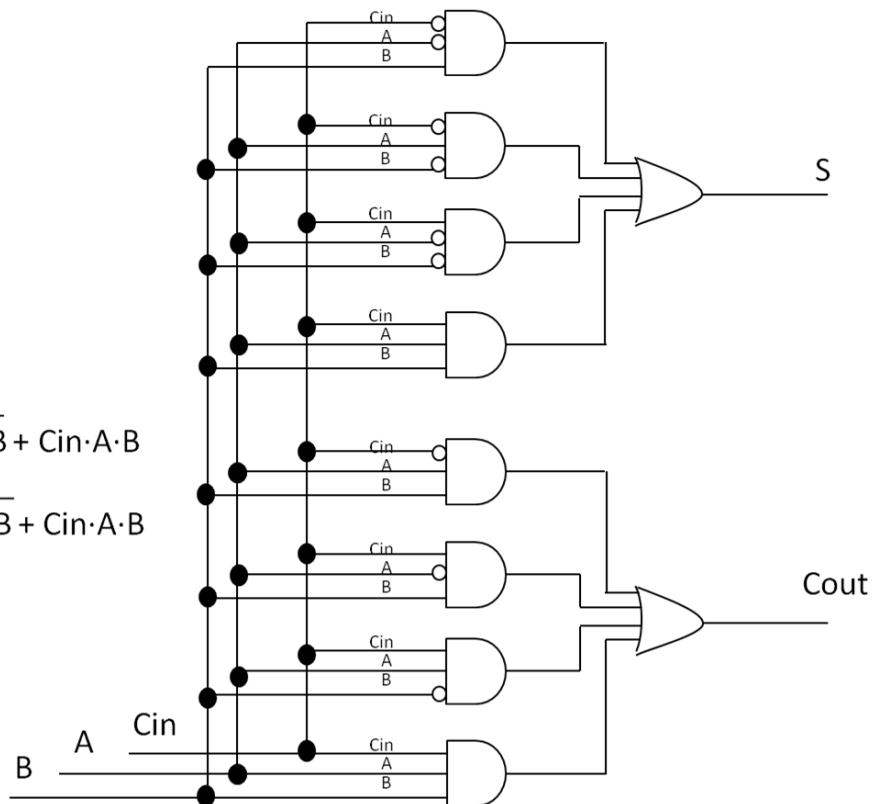
En base a eso pueden dibujarse los circuitos lógicos internos del sumador:

Por ejemplo, en forma normal disyuntiva:

$$S = \overline{Cin} \cdot \overline{A} \cdot B + \overline{Cin} \cdot A \cdot \overline{B} + Cin \cdot \overline{A} \cdot \overline{B} + Cin \cdot A \cdot B$$

$$\text{Count} = \overline{Cin} \cdot \overline{A} \cdot B + \overline{Cin} \cdot A \cdot \overline{B} + Cin \cdot \overline{A} \cdot \overline{B} + Cin \cdot A \cdot B$$

Donde A y B son un bit de cada sumando, Cin es el carry proveniente de la anterior suma, S es la suma, y Cout es el posible carry

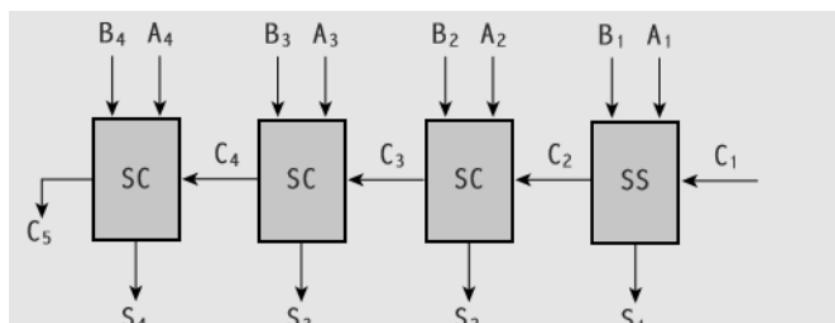


CIRCUITO SUMADOR-BINARIO EN PARALELO

Ahora analizaremos cómo a partir de la expresión de una función booleana se puede confeccionar el circuito que lleve a cabo la operación que describe. Supongamos que queremos sumar dos números de 4 bits cada uno:

$$\begin{array}{r} A \\ + B \\ \hline \end{array}$$

$$\begin{array}{r} 0101 \\ + 1010 \\ \hline \end{array}$$

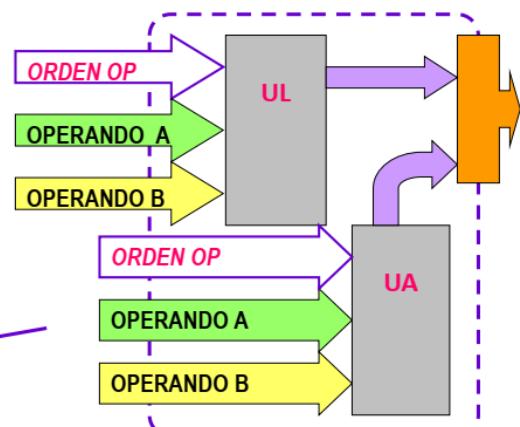


UNIDAD 5: ÁLGEBRA DE BOOLE

CONCEPTO DE UAL

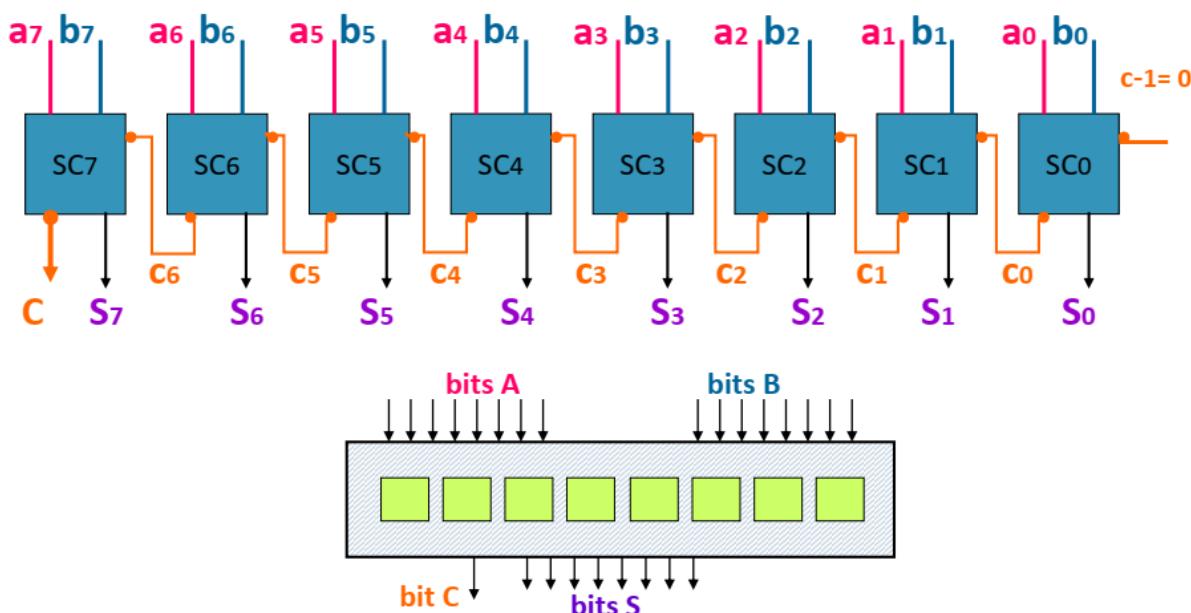
La Unidad Aritmética y Lógica es el componente de la CPU que realiza las operaciones lógicas (AND, OR, XOR, etc.) y aritméticas (en principio suma y resta).

Podemos pensar entonces en una estructura con dos operadores básicos, uno lógico –**UL** y uno aritmético –**UA**, como muestra la figura adjunta



Como se observa en esta figura, los operadores reciben los operandos (datos) y una orden de operación (para indicar QUE operación deben hacer).

SUMADOR DE 8 BITS



UNIDAD 5: ÁLGEBRA DE BOOLE

LA RESTA EN LA UAL

La resta se implementa utilizando el mismo dispositivo sumador, gracias a la propiedad de la notación posicional de cantidades denominada COMPLEMENTO.

Para un número N de “p” dígitos expresado en base “b” se definen:

COMPLEMENTO DIRECTO (a la base menos uno)

$$CD(N) = (b^p - 1) - N$$

COMPLEMENTO AUTÉNTICO (a la base)

$$CA(N) = (b^p) - N$$

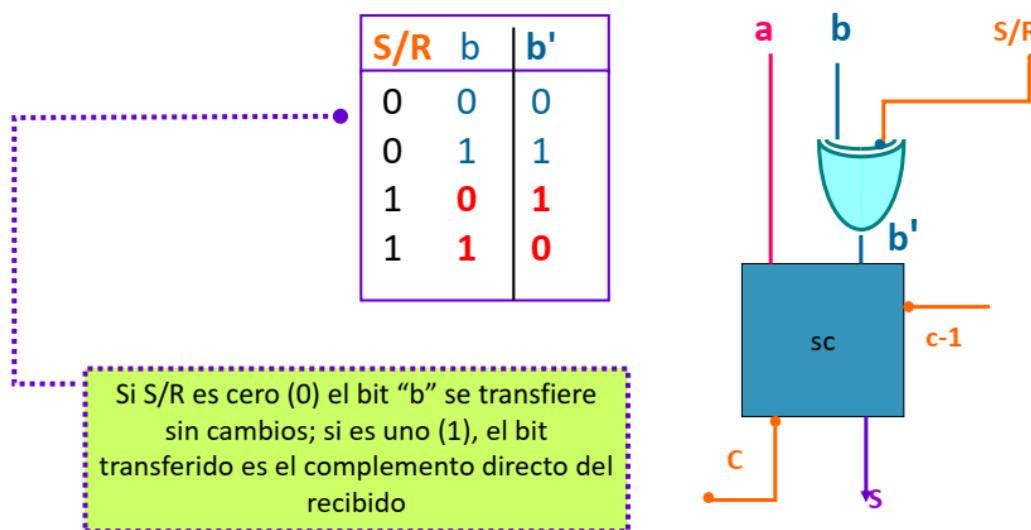
Y de allí

$$CA(N) = CD(N) + 1$$

Según se ha visto, la resta puede obtenerse “sumando” al minuendo el complemento auténtico del sustraendo.

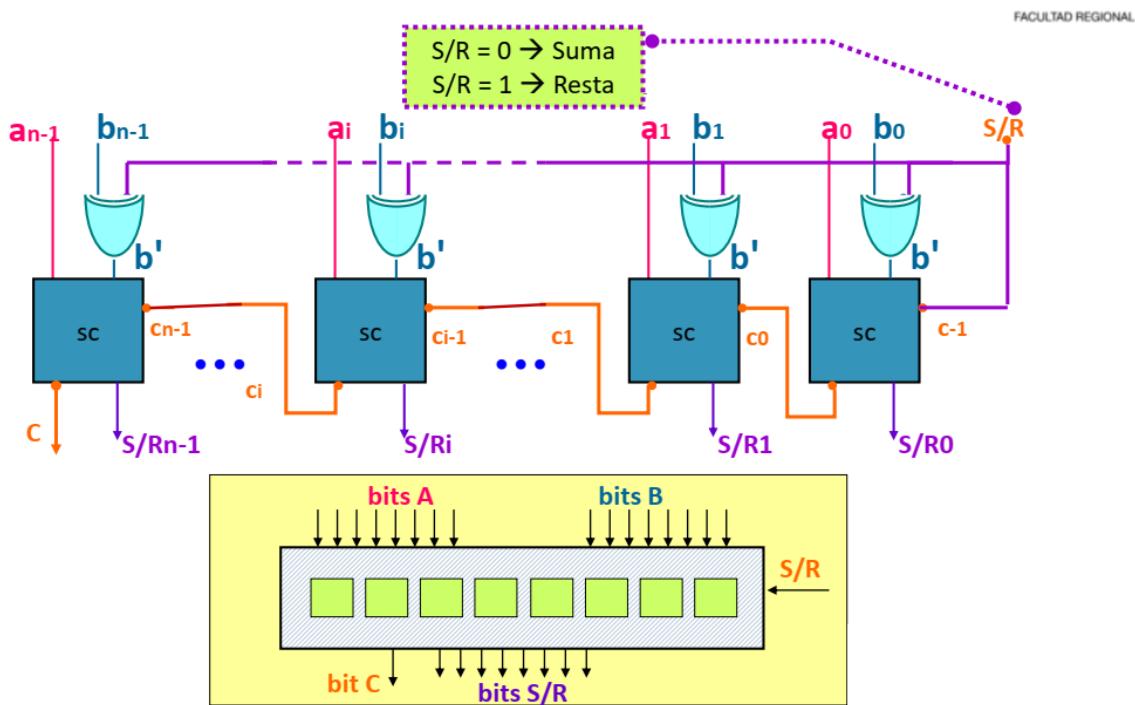
En binario el complemento directo se obtiene invirtiendo todos los bits del registro. Por lo tanto, es sencillo deducir el esquema del sumador-restador elemental que realice las operaciones ($a + b$) y ($a - b$), según se indique.

SUMADOR – RESTADOR EN COMPLEMENTO AUTÉNTICO

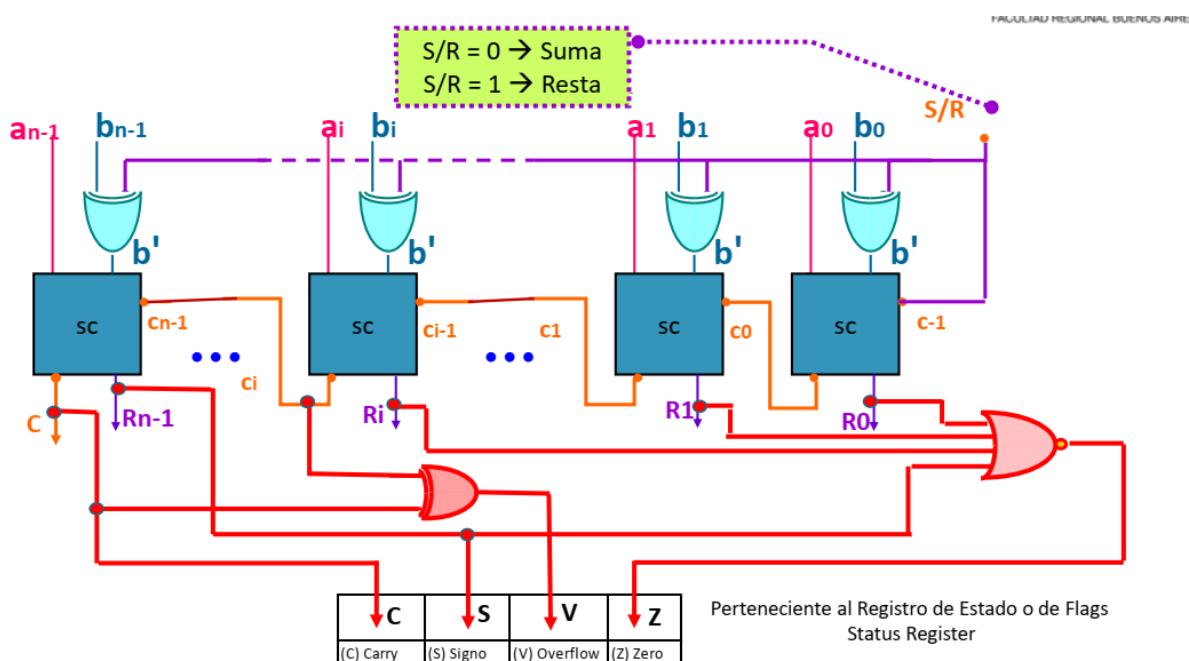


UNIDAD 5: ÁLGEBRA DE BOOLE

SUMADOR-RESTADOR DE N BITS



SUMADOR-RESTADOR DE N BITS CON SUS FLAGS



UNIDAD 6: LÓGICA DIGITAL

INTRODUCCIÓN

Las **Expresiones booleanas** permiten la representación algebraica del valor de funciones, que representan las salidas de un circuito.

En los **circuitos lógicos** de sistemas digitales los estados 0 o 1 se determinan por niveles de tensión.

Representaremos los circuitos en base a las compuertas vistas.

Sabemos que cada compuerta en sí constituye un circuito electrónico desarrollado para cumplir la función para el cuál fue diseñada.

CIRCUITOS COMBINACIONALES

Los circuitos lógicos se clasifican en dos:

Combinacionales: En los circuitos combinacionales, las salidas siempre van a depender solamente de la combinación de las variables de entrada.

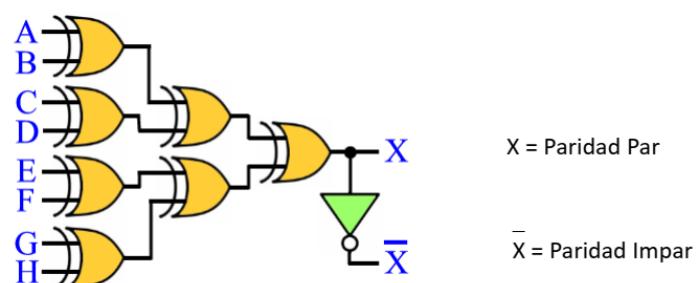
Secuenciales: En los circuitos secuenciales, a diferencia de los combinacionales, la salida no solo dependerá de las variables de entrada, sino también dependerá del estado en que se encontraba su salida. Es por eso que a estos circuitos también se los conoce como circuitos con memoria, circuitos biestables o simplemente **FLIP FLOP**.

Circuito generador de paridad: Un circuito generador de paridad permite la detección de error en la transmisión de información entre un emisor y un receptor de información binaria cuyo coeficiente de probabilidad de error sea menor o igual a un bit.

Un bit de paridad se genera mediante un circuito sencillo compuesto por puertas XOR.

Un generador para un circuito de 8 bits deberá cumplir la función:

$$BP = [(a \oplus b) \oplus (c \oplus d) \oplus (e \oplus f) \oplus (g \oplus h)]$$



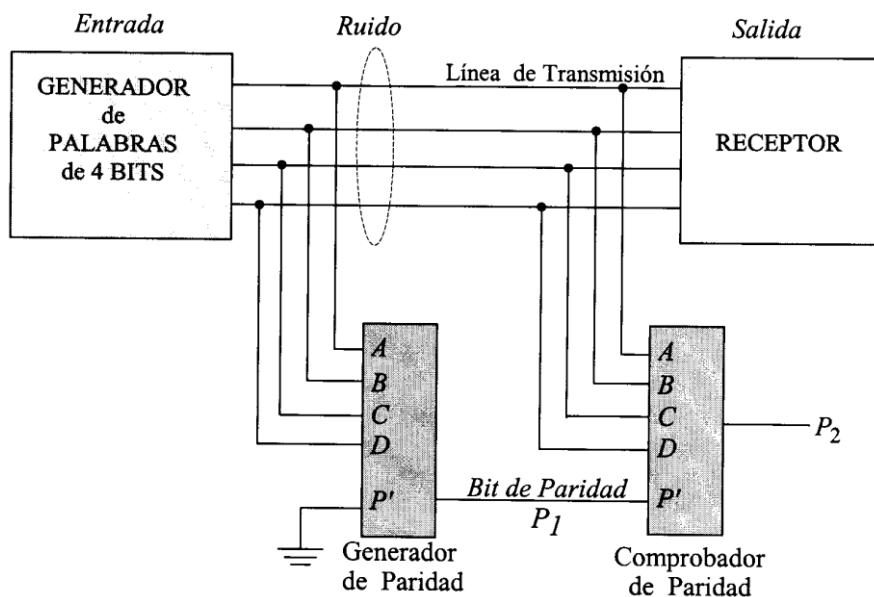
Paridad PAR: El número de unos que contamos, INCLUIDO EL DE PARIDAD, debe ser PAR.

Paridad IMPAR: El número de unos que contamos, INCLUIDO EL DE PARIDAD, debe ser impar.

UNIDAD 6: LÓGICA DIGITAL

CIRCUITOS COMBINACIONALES

Circuito generador de paridad:



Circuito comparador de magnitud: Es común que se necesite comparar dos números binarios de n bits para saber si éstos son iguales, o si uno es mayor o menor que otro. Podemos simplificar el problema analizando la comparación de dos bits.

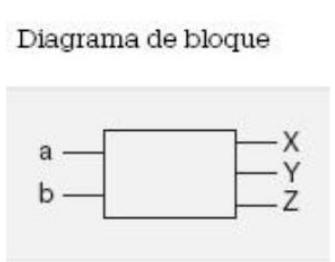
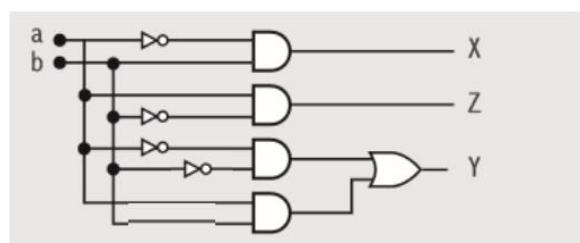


Tabla de verdad

a	b	$a > b$	$a < b$	$a = b$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1



Circuito decodificador de código: Existe otro tipo de decodificadores que permiten señalar en una de sus salidas qué combinación binaria se dio en la entrada. Los decodificadores de este tipo se denominan " $nX2^n$ " (se lee " n x 2 a la n " o " n a 2 a la n ") y contemplan una salida activa para cada posible combinación de entradas.

UNIDAD 6: LÓGICA DIGITAL

CIRCUITOS COMBINACIONALES

Circuito decodificador de código:

Diagrama de bloque sin entrada de habilitación

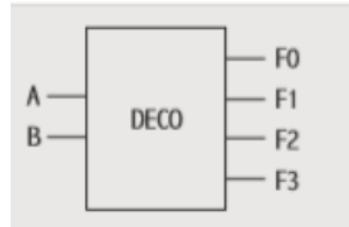
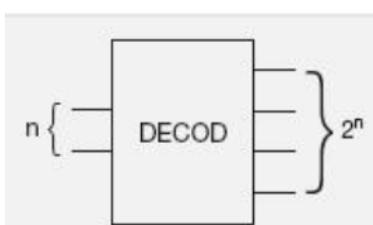
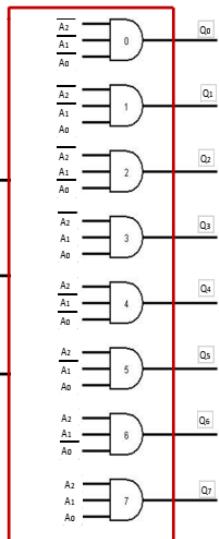


Diagrama de bloque del decodificador $n \cdot 2^n = 2 \cdot 2^2 = 2 \cdot 4$ sin entrada de habilitación

A2	A1	A0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

A₀ _____
A₁ _____
A₂ _____



Una de las utilidades principales es dentro de la arquitectura de las memorias, direccionar la locación a acceder.

A Entradas
 $Q = 2^A$ Salidas

2^A Compuertas And con A entradas cada una

Cada compuerta AND tiene el minitérmino correspondiente que activa la salida según la tabla de verdad.

Para cada combinación de las variables de entrada solo una estará activa correspondiente al minitérmino dado por la combinación de entrada

Memorias ROM (memorias de sólo de lectura): El arreglo de compuertas AND-OR, programado de esta manera, es un ejemplo simple de memoria ROM (Read Only Memory). Desde el punto de vista de los circuitos lógicos, forma parte de los dispositivos de lógica programable y se caracterizan por tener conexiones fijas en el arreglo de compuertas AND y conexiones programables en el arreglo de compuertas OR.

Una vez establecida la combinación binaria de las salidas, éstas se mantienen inalterables.

CIRCUITOS COMBINACIONALES

Memorias sólo de lectura

Distribución de líneas en una memoria de sólo lectura:

La distribución de una memoria forma una matriz de $M \times N$ bits (M direcciones de N bits cada una).

El bus de direcciones tiene A líneas, tal que $2^A = M$.

El bus de datos tiene N líneas para transferir los N bits leídos.

El bus de control tiene una línea que habilita el chip y normalmente se denomina CS (Chip Select).

En cualquiera de los ROM, la red AND es inalterable y la red OR es programable.

Diagrama de bloque de cualquiera de las memorias de sólo lectura.

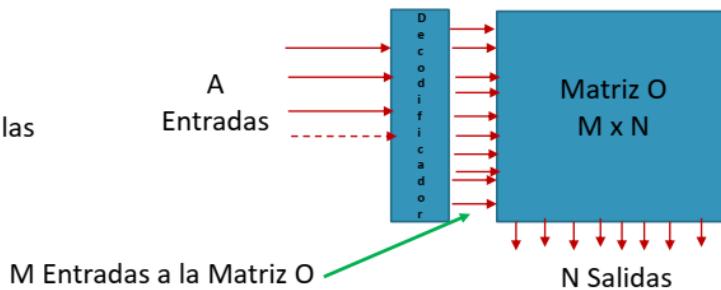
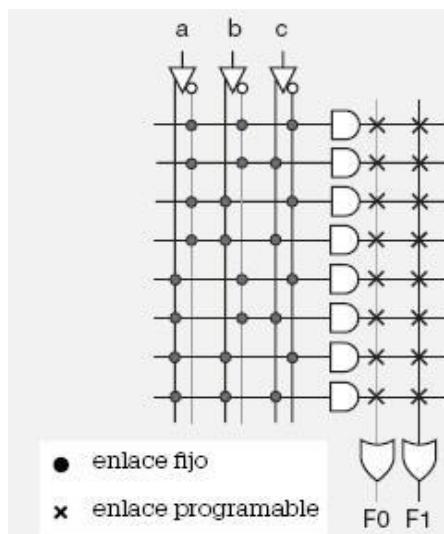


Diagrama de lógica de una ROM sin programar: Una memoria ROM no es más que otra forma de representar los miniterminos para cada función de A variables; así, en el diagrama siguiente se representan las ocho combinaciones posibles de tres variables (a, b y c) como enlaces fijos en una red AND que, acoplada a una red OR, podrá “programarse” para cada una de las dos funciones de salida F_0 y F_1

La arquitectura de una Memoria ROM por lo tanto está conformada por un Decodificador y una Matriz, denominada Matriz O

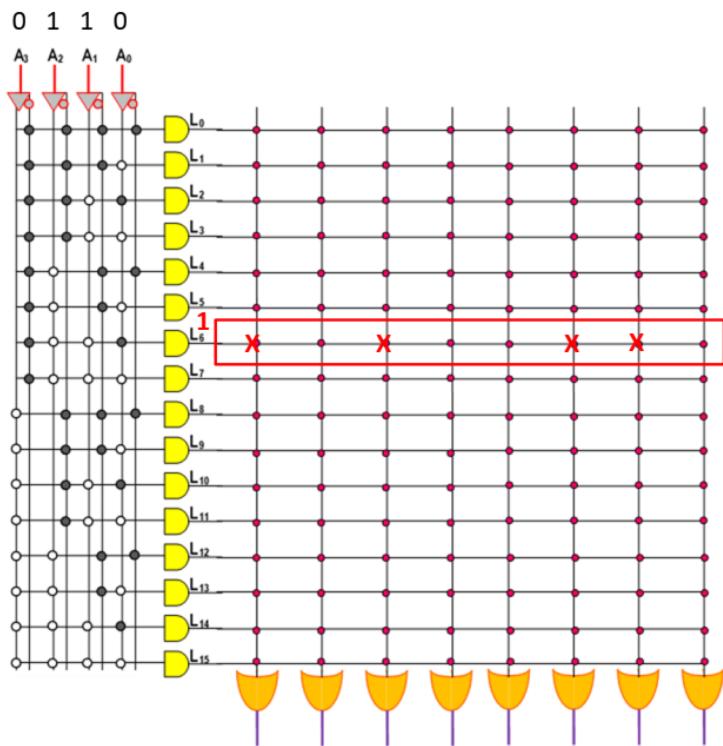


UNIDAD 6: LÓGICA DIGITAL

CIRCUITOS COMBINACIONALES

Memorias sólo de lectura

Diagrama de lógica de una ROM sin programar:



Dirección	Memoria Principal
A ₃ A ₂ A ₁ A ₀	
0 0 0 0	
0 0 0 1	
0 0 1 0	
0 0 1 1	
0 1 0 0	
0 1 0 1	
0 1 1 0	0 1 0 1 1 0 0 1
0 1 1 1	
1 0 0 0	
1 0 0 1	
1 0 1 0	
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

Contenido

Familias de memoria ROM:

PROM = Programmable ROM: Es aquella Memoria que en su estado original contiene en todas sus ubicaciones contenidos de unos (1), de forma tal que permite ser grabada una vez, mediante un programador de PROM, quedando los datos grabados en forma permanente y luego cumpliendo la función de una memoria ROM. No se la puede volver a grabar.

EPROM = Erasable Programable ROM: La misma permite ser borrada a través de luz ultravioleta para convertirse en una PROM y poder volver a ser grabada. Luego su funcionamiento es el de la memoria ROM.

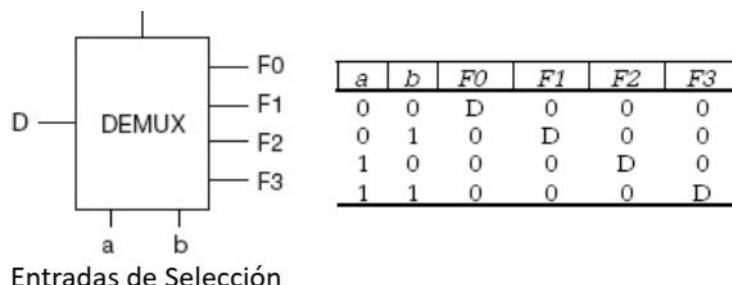
EEPROM = Electrically Erasable Programable ROM: A diferencia de la EPROM, la misma puede ser borrada y reprogramada eléctricamente.

UNIDAD 6: LÓGICA DIGITAL

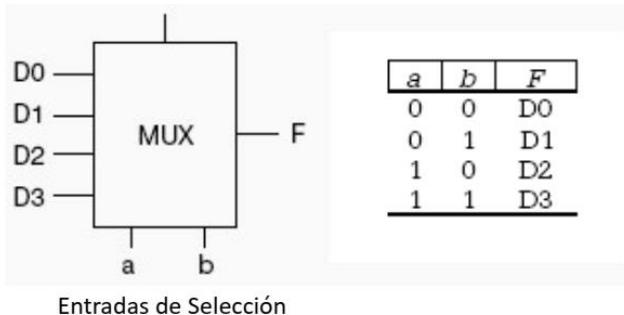
CIRCUITOS COMBINACIONALES

Circuitos multiplexores y demultiplexores:

Demultiplexor: Es un circuito cuya función digital es “encaminar” la información que viene en una sola línea de entrada, a 1 de las 2^E salidas, donde E son las llamadas Entradas de Selección que justamente es seleccionar la salida por la cual se encaminará la información de entrada. Algunas de las aplicaciones de un demultiplexor son, por ejemplo, distribuir una entrada de datos en serie a una salida en paralelo o transferir bits de una línea de bus a un registro determinado; por eso también se lo denomina distribuidor.



Multiplexor: Es un circuito combinacional cuya función digital es “encaminar” las señales binarias de 1 de 2^E líneas posibles de entrada en una única línea de salida. La línea de entrada de dato se “elige” a partir de los valores que puedan tomar las n líneas de selección.

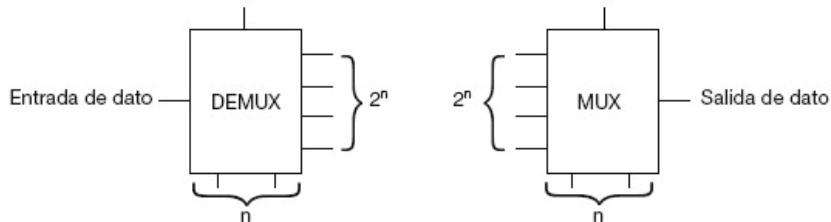


Aplicaciones de los multiplexores

1. Un MUX se puede implementar con cierto número de compuertas lógicas, porque básicamente es un decodificador con sus salidas asociadas a una única compuerta.
2. Los MUX de pocas entradas se utilizan en los dispositivos de lógica programables descritos más abajo.
3. Asimismo, el MUX se utiliza para la conversión de bits transferidos en paralelo a serie, esto es, de a uno por vez, utilizando una sola línea.
4. La multiplexación de bits de dirección en memorias se emplea, por ejemplo, cuando un bloque de caché supera la capacidad del bus, si el bloque es de 128 bits y el bus de 64; entonces, primero se transfiere un grupo de 64 bits y luego el otro.
5. Se puede armar un módulo desplazador que permite mover bits a derecha e izquierda en registros de desplazamiento bidireccionales, bajo microoperaciones de control, por ejemplo, las que genera la unidad de control cuando se ejecuta una instrucción de desplazamiento (ver sección "Registros con facilidad de desplazamiento").
6. Se emplean en la arquitectura interna de un Registro como veremos más adelante.

CIRCUITOS COMBINACIONALES

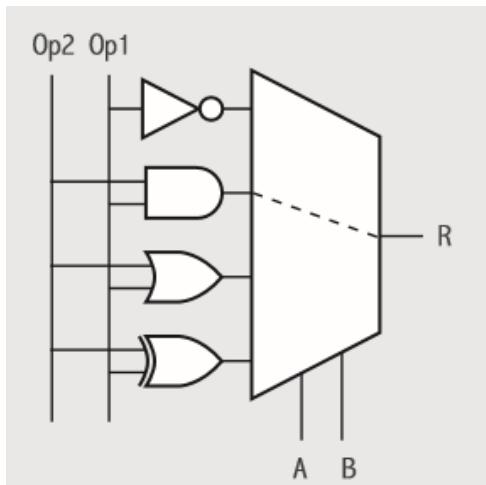
Circuitos multiplexores y demultiplexores (resumen)



Ejemplo de una unidad aritmético-lógica (ALU) para 4 operaciones de tipo lógico utilizando un multiplexor.

Supongamos las operaciones NOT (negación o complemento), AND (conjunction o producto lógico), OR (disyunción inclusiva o suma lógica), XOR (disyunción excluyente o suma exclusiva), que identificamos con los números 00, 01, 10, 11, respectivamente. Las señales de control indican cuál de las "instrucciones" pasan por la línea de resultado, al operar los bits Op1 y Op2. Para este ejemplo, A = 0 y B = 1, entonces, el resultado R es el producto lógico entre ambos operadores:

Op1 AND Op2.



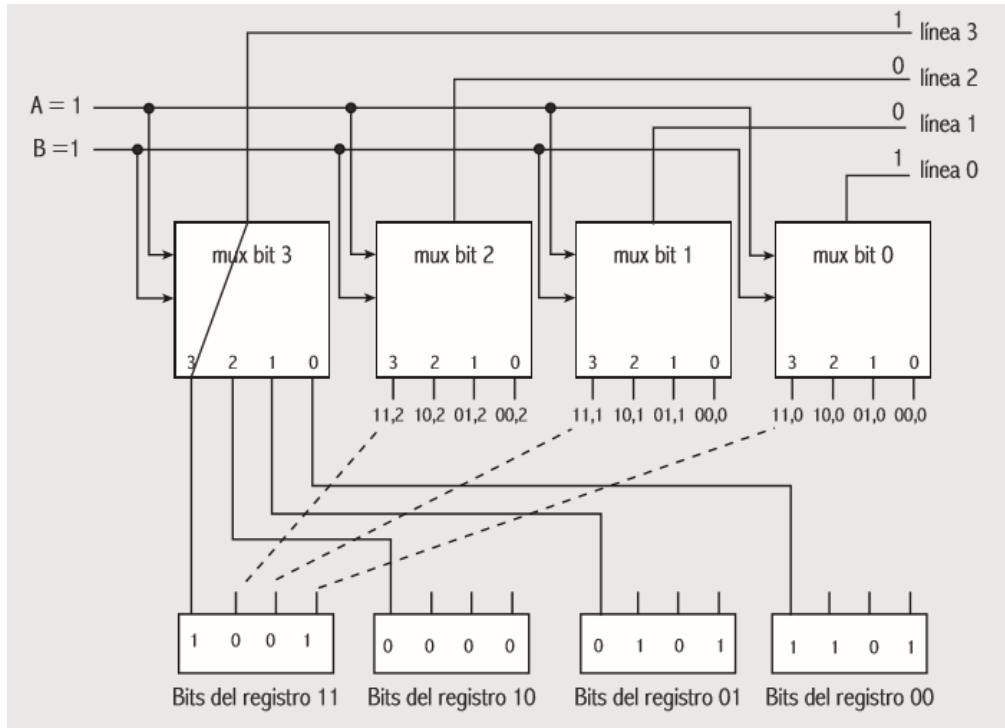
Bus asociado a un multiplexor-demultiplexor

Un bus es un conjunto de conductores que permiten relacionar registros que actúan de emisores o de receptores de bits. En este caso se indica que los registros están asociados “a un bus común o bus único”.

Cada multiplexor habilita el paso de una señal sobre el bus comandado por la dirección en sus entradas de selección. Así: A = 1 y B = 1 indican que D3 es la línea de entrada seleccionada para salir por F.

CIRCUITOS COMBINACIONALES

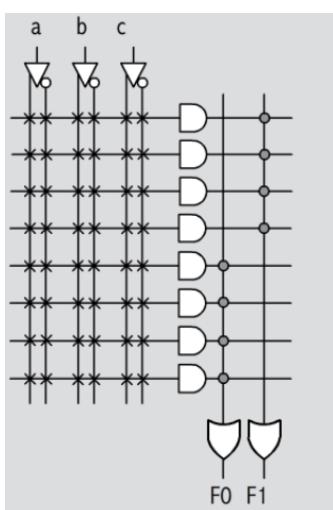
Bus asociado a un multiplexor-demultiplexor



Otro tipo de Memorias sólo de lectura

Dispositivos tipo PAL

En una nueva categoría de dispositivos de lógica programable, se encuentran los denominados PAL (Programmable Array Logic), más flexibles y rentables que las típicas ROM. Su función es similar a la de la ROM, pero en este caso se invierten los papeles de las redes ANDOR. La red AND es programable y la OR es fija. El diagrama para el ejemplo anterior puede ser:



UNIDAD 6: LÓGICA DIGITAL

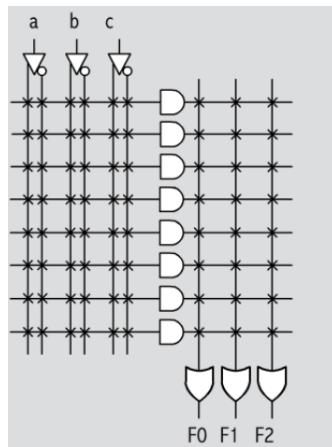
CIRCUITOS COMBINACIONALES

Otro tipo de Memorias sólo de lectura

Dispositivos tipo PLA o F-PLA (field-PLA)

Los dispositivos tipo F-PLA (field = campo) constituyen otra categoría de los dispositivos de lógica programable, y tienen aún mayor flexibilidad, dado que ambas redes son programables.

Diagrama de lógica para un PLA no programado, esta vez de tres entradas y tres salidas: F0, F1 y F2



OTROS CIRCUITOS COMBINACIONALES

Circuito convertidor de código

Son circuitos cuya función digital es obtener en las salidas combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricicos).

Las entradas pueden ser señales que provienen de algún dispositivo de entrada de información (p. ej., señales de teclado) o pueden ser combinaciones binarias de otro código para convertir.

Ejemplo: Un circuito que convierte un dígito BCD puro en su correspondiente valor BCD 2421.

Entradas BCD				Salidas BCD			
8	4	2	1	2	4	2	1
A0	A1	A2	A3	F0	F1	F2	F3
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

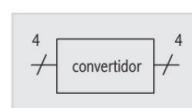


Fig. 6.6. Diagrama de bloque.

UNIDAD 6: LÓGICA DIGITAL

OTROS CIRCUITOS COMBINACIONALES

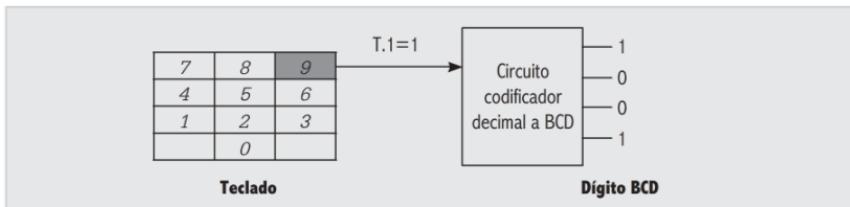
Circuito codificador

Veremos cómo las señales binarias provenientes de un teclado numérico pueden transformarse en salidas codificadas, por ejemplo, en BCD. Consideraremos teclas numéricas sin tener en cuenta las teclas de operación (+, -, *, etc.) comunes en todo teclado.

Uno de los códigos más usados en la representación de números es el BCD 8421 o BCD puro. La tabla de verdad muestra sólo 10 de las $2^{10} = 1024$ combinaciones posibles para 10 entradas, porque se considera que no se presionan dos teclas en forma simultánea.

Tabla 6-4. Tabla de verdad

Entradas que identifican las teclas											Salidas BCD			
T9	T8	T7	T6	T5	T4	T3	T2	T1	TO	F ₃	F ₂	F ₁	F ₀	
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	1



Circuito decodificador a 7 segmentos

Opera en forma inversa al codificador en el siguiente sentido: la información que ingresa en el circuito implica combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricos) y las convierte a señales binarias para representar en algún dispositivo de salida de información.

