

3

Representación de datos en la computadora

Contenido

3.1 Introducción	50
3.2 Flujo de datos dentro de una computadora	50
3.3 Códigos de representación de caracteres alfanuméricos	53
3.4 Códigos de representación decimal (BCD)	56
3.5 Códigos de representación numérica no decimal.....	59
3.6 Representaciones redundantes	68
3.7 Resumen.....	71
3.8 Ejercicios propuestos.....	71
3.9 Contenido de la página Web de apoyo.....	72

Objetivos

- Reconocer los códigos de representación de caracteres (alfanuméricicos y numéricos).
- Reconocer los formatos numéricos: Representaciones decimales, representaciones numéricas para números enteros o para números reales.
- Reconocer sólo algunas representaciones redundantes que permiten la detección y corrección de errores.
- Comprender los convenios de representación en Punto Flotante Exceso IEEE-P754.



En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.



Unicode: acrónimo de *Universal Code* que permite representar 216 combinaciones distintas, es decir 65.536 símbolos.

3.1 Introducción

En el desarrollo de este capítulo se describe la representación interna de los formatos de “tipos de datos” que los lenguajes de programación permiten declarar, para las distintas estructuras de dato. Por ejemplo: una variable *string* se almacenará en un código de representación alfanumérico, como UNICODE o ASCII; una variable *unsigned integer*, como un número entero no signado, conocido como “de coma fija” o “de punto fijo”; un número como el $7,5_{(10)}$ se almacenará como un número de “punto flotante” o “coma flotante”. La mayoría de ellos se ajusta a convenios estandarizados, de los cuales en este capítulo se consideraron los más difundidos en los diseños de las computadoras actuales. Todos pasamos por la experiencia de comprar un teclado para nuestra computadora personal; un comprador no especializado desconoce que los teclados de distintas marcas responden a un estándar que independiza la compra del dispositivo respecto de la marca del fabricante de su propia computadora, ya que el ingreso de datos por teclado responde al estándar de representación de códigos alfanuméricos.

En cuanto a datos “operables”, los especialistas del área deben conocer cuál es la diferencia entre la representación de un entero y la de una fracción o de un número real.

En esta introducción hacemos referencia a que utilizaremos los términos “coma fija” o “punto fijo” en forma indistinta en relación con el término en inglés *fixed-point*; de la misma manera se utilizarán los términos “coma flotante” o “punto flotante” respecto de las representaciones *floating-point*. Esto se debe a que la “coma” se utiliza en nuestro sistema numérico como el separador de decimales, y en el caso de la palabra “punto” corresponde a la traducción literal del término en inglés *point*. Se considera que ambos son aceptables, dado que aparecen de manera indistinta en la bibliografía consultada.

3.2 Flujo de datos dentro de una computadora

Una computadora está constituida por una memoria, que almacena programas y datos en forma temporal, y un órgano ejecutor de estos programas conocido como unidad central de proceso (CPU).

Si se la analiza desde un ejemplo más cotidiano, pero práctico, la memoria cumple la función de una hoja en la que se escribieron los pasos para la resolución de un problema y los datos que intervienen en los cálculos y donde se volcarán los resultados obtenidos. A su vez, usted interpreta los pasos, toma los datos y los vuelve en una calculadora de bolsillo. Dentro de la computadora, quien realiza esta actividad es la unidad de control (CU) y la de la calculadora de bolsillo es la unidad aritmético-lógica (ALU). Así, el conjunto CU-ALU constituye la ya mencionada CPU.

Las variaciones en el diseño interno de las distintas computadoras actuales no modifican este esquema global, pero si afectan el procesamiento de los programas y los datos que deben transformarse en el interior para adaptarse a la estructura de cada computadora. Sin embargo, el programador no es el responsable de implementar estas transformaciones; por ejemplo, la compilación de un programa en lenguaje simbólico permite adaptar su estructura y generar el programa en el código de máquina que la computadora puede entender. Entonces hay un “mediador” entre los programas y los datos que vienen “de afuera” y las necesidades internas para procesarlos. Este vínculo que pasa de lo estándar a lo particular en cada equipo es una rutina de conversión enlazada al código del programa.

En resumen, tanto las instrucciones de un programa como los datos que ingresan en la memoria desde el exterior lo hacen en un código alfanumérico de representación de caracteres, por ejemplo, el ASCII. Las instrucciones son transformadas por un programa de traducción (que puede ser un ensamblador, un traductor o un compilador) a código de máquina, que es el que entiende el procesador, para luego ejecutarse. Los encargados de la transformación de los datos, respetando el formato definido en el programa, suelen ser rutinas previstas por

bibliotecas estándar del lenguaje e incluidas en el programa. Por lo tanto, aunque los datos ingresen desde el teclado en un código alfanumérico de caracteres, éstos se almacenarán en la memoria de distintas formas, según los requerimientos determinados por la declaración de los datos del programa; éste es el único que puede reconocer si el byte de memoria accedido es, por ejemplo, un operando o un carácter alfanumérico (fig. 3.1).

Por otra parte, una computadora necesita comunicarse con el medio externo por medio de dispositivos de entrada o salida, y éstos se diseñan para conectarlos a una amplia gama de computadoras diferentes entre sí, lo que induce a pensar que los periféricos utilizan códigos estándar para la representación de información. La forma de ingreso o egreso de datos y programas en una computadora respeta estos códigos de representación de caracteres que abastecen las necesidades de **intercambio** de información a nivel general. Esto es independiente de las transformaciones que se realizan para su tratamiento interno.

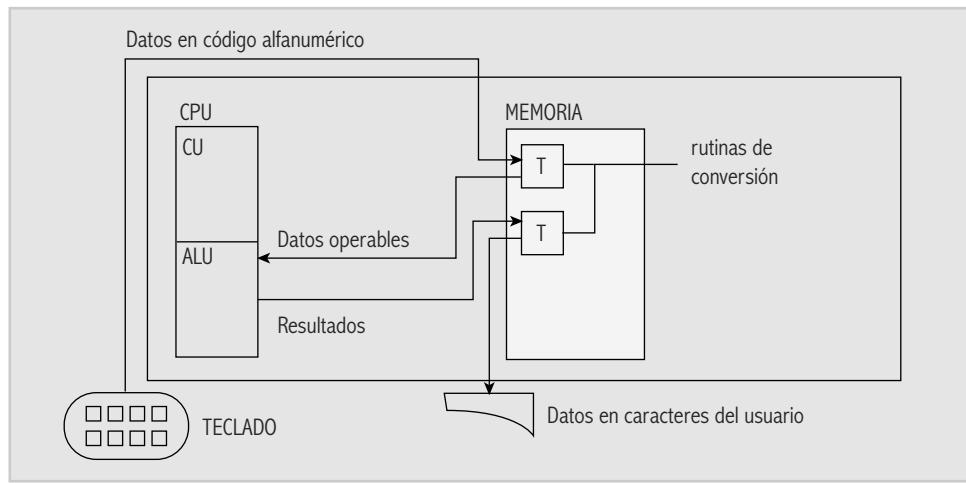
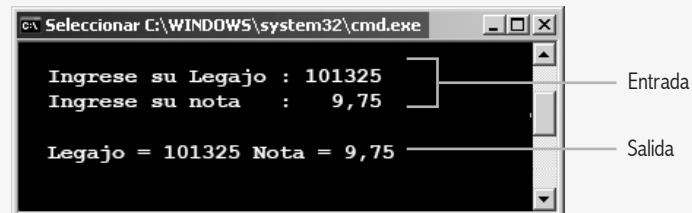


Fig. 3.1. Conversión de los datos durante el procesamiento.

Para ilustrar el flujo de datos utilizaremos un programa que permite ingresar dos datos de diferentes tipos: uno entero y uno real, utilizando la especificación de formato en lenguaje C. La función `scanf` permite el ingreso de los datos, su sintaxis es la siguiente: `scanf ("Formato", &identificación de la variable)`. `scanf ("%d", &x);` permite el almacenamiento de un número entero en la variable x, `scanf ("%f", &m);` permite el almacenamiento de un número en coma flotante en la variable m. La variable representa un lugar en memoria en la que se almacenará un valor que el usuario ingresará por el teclado. Luego de cada ingreso se presiona la tecla Enter.

```
#include< stdio.h >; //incluye la biblioteca estándar que contiene a scanf y a printf
void main()
{
// Definición de la variables
int legajo;
float nota;
clrscr();
//Ingreso de Datos con función scanf
printf("Ingrese su legajo:");
scanf("%d",&legajo);
printf("Ingrese su nota con dos decimales:");
scanf("%f",&nota);
// Salida de datos ingresados con función fprintf
printf("\n\n Legajo=%d Nota=%g",legajo, nota);
}
```



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, con 7 bits se obtienen $2^7 = 128$ combinaciones distintas, que son suficientes para asignar un símbolo alfabético, numérico o especial a cada una de ellas.

Si bien se puede establecer infinidad de códigos alfanuméricos de “entendimiento” entre el usuario y las distintas computadoras, se desarrollaron convenios de uso internacional, cuya finalidad es concretar la compatibilidad entre dispositivos de distinto origen.

Cada carácter emitirá una serie de señales binarias al “digitarse” sobre el teclado, que quedará alojada internamente como una combinación de bits, de acuerdo con alguna representación de caracteres alfanuméricos preestablecida.

Supóngase que se ingresa el dato alfanuméricico “+105”, el proceso de codificación hará que éste quede registrado en la memoria, por ejemplo, en código ASCII, como se muestra en la figura 3.2.

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

Fig. 3.2. Símbolos del usuario y su representación binaria en ASCII

Formato de una entidad binaria: es la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento en la computadora.

Se denomina **formato** de una entidad binaria a la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento en la computadora. Los formatos pueden ser de longitud fija o variable, según la categoría de dato que represente. Es importante destacar que el que interpreta un grupo de bits bajo un formato determinado es un programa. Esto significa que una cadena de bits en un registro de la ALU puede ser interpretada por el programa que la utiliza como operando, como un binario sin signo o con él, o simplemente puede representar una parte del operando en sí, como la mantisa de un operando definido como real. Del mismo modo, un byte alojado en la memoria puede ser un carácter alfanumérico, un operando o incluso una instrucción, según quién lo “interprete”.

En la tabla 3-1 se detallan distintos tipos de representaciones de datos que una computadora puede manejar y que más adelante se explicarán. Debe quedar claro que, en general, del conjunto de posibilidades para cada una de las representaciones, sólo se utilizará una de ellas. Por ejemplo, para una computadora las representaciones alfanuméricas siempre se interpretarán en código ASCII, las representaciones decimales se manejarán en BCD puro, los números enteros se expresarán en complemento a la base y los números reales utilizarán una representación en IEEE P754. De manera independiente, la misma computadora puede utilizar formatos de números enteros en punto fijo sin signo o con él, ya que ello depende de la definición del tipo de dato que el programa de usuario necesite.

Tabla 3-1. Distintos tipos de representaciones 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 (deteccción de errores)	Códigos de paridad	Paridad vertical o a nivel carácter Paridad vertical o a nivel bloque Paridad entrelazada Código de Hamming

3.3 Códigos de representación de caracteres alfanuméricicos

Si bien una computadora es una “calculadora aritmética” por excelencia, gran parte de sus aplicaciones consiste en el tratamiento de cadenas de caracteres llamadas *strings*. Un *string* es una unidad formada por caracteres representados según un código alfanumérico preestablecido para esa computadora. Uno de los códigos de representación de caracteres más usado es el “ASCII”.



El código ASCII de 7 bits permite determinar $2^7 = 128$ combinaciones posibles que representan 128 símbolos distintos.

3.3.1 Código ASCII

El código **ASCII** de 7 bits (*American Standard Code for Information Interchange* o Código Estándar Americano para Intercambio de Información) representa cada carácter con 7 bits, que permite determinar $2^7 = 128$ combinaciones distintas, suficientes para establecer una única relación carácter-combinación binaria, que se representa en la tabla 3-2.



Encuentre un simulador sobre codificación ASCII en la página Web de apoyo.

Tabla 3-2. Tabla de código ASCII (de 7 bits).

HEX. N°	0	1	2	3	4	5	6	7
HEX. N°	BITS 654 3210	000	001	010	011	100	101	110
0	0000	NUL	DLE	SP	0	@	P	`
1	0001	SOH	DCI	!	1	A	Q	a
2	0010	STX	DC2	"	2	B	R	b
3	0011	ETX	DC3	#	3	C	S	c
4	0100	EOT	DC4	\$	4	D	T	d
5	0101	ENQ	NAK	%	5	E	U	e
6	0110	ACK	SYN	&	6	F	V	f
7	0111	BEL	ETB	'	7	G	W	g
8	1000	BS	CAN	(8	H	X	h
9	1001	HT	EM)	9	I	Y	i
A	1010	LF	SUB	*	:	J	Z	j
B	1011	VT	ESC	+	;	K	[k
C	1100	FF	FS	,	<	L	\	l
D	1101	CR	GS	-	=	M]	m
E	1110	SO	RS	.	>	N	^	n
F	1111	SI	US	/	?	O	-	o
							DEL	

<i>Significado en inglés de los caracteres de control de las columnas 0 y 1</i>		
NUL, null	VT, vertical tabulation	SYN, synchronous idle
SOH, start of heading	FF, form feed	ETB, end of transmission block
STX, start of text	CR, carriage return	CAN, cancel
ETX, end of text	SO, shift out	EM, end of medium
EOT, end of transmission	SI, shift in	SUB, substitute
ENQ, enquiry	DEL, data link escape	ESC, escape
ACK, acknowledge	DC1, device control 1	FS, file separator
BEL, bell (audible signal)	DC2, device control 2	GS, group separator
BS, backspace	DC3, device control 3	RS, record separator
HT, horizontal tabulation (punched card skip)	DC4, device control 4	US, unit separator
LF, line feed	NAK, negative acknowledge	DEL, delete

Este patrón se puede definir como “oficial” y en todos los textos en los que se incluya esta tabla no habrá variaciones que den lugar a pensar que “hay varios ASCII”.

Los primeros 3 bits (6, 5 y 4) de mayor significación en cada combinación se denominan bits de **zona** y en la tabla de doble entrada numeran las columnas 0 a 7.

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

Los primeros 32 caracteres pueden parecer “raros”, sin embargo, son de gran utilidad porque permiten **cierto control** de las actividades de la computadora. Por ejemplo, para tabular hay una tecla o dos en el teclado que permiten “ocupar” un lugar en la pantalla con la combinación 0000 1001; aunque no se visualice, “marcan” ese lugar para que el cursor salte allí. El carácter BEL permite emitir un sonido que puede utilizarse como aviso al operador de la computadora. Los códigos SO y SI, shift activo o inactivo, se utilizan para ordenar a la impresora que imprima caracteres estándar o condensados. El resto de los caracteres es familiar para el usuario: Letras mayúsculas y minúsculas, dígitos, signos de puntuación, etcétera. Sin embargo, hay ciertas cuestiones para tener en cuenta. Por un lado, las mayúsculas se consideran distintas a las minúsculas, aun cuando se trate de la misma letra, debido a que su forma de representación **simbólica** es diferente por completo (A vs. a). Las combinaciones que representan letras están numéricamente ordenadas, esto es, la combinación de “A” es menor que la de “B”, lo que permite ordenar alfabéticamente un conjunto de datos. Por último, el espacio en blanco está definido por la combinación 0010 0000 porque el blanco o el espacio “ocupan lugar aunque no se vean”.

Para terminar, es importante saber que la combinación binaria que representa un carácter también se expresa en sus formas hexadecimal y decimal de la siguiente forma:

$$\text{“A”} = 100\ 0001 = 41h = 65d \quad h = \text{hexadecimal} \quad d = \text{decimal}$$

3.3.2 Código ASCII ampliado

Si bien el ASCII se concibió como código de 7bits, su uso generalizado dio lugar a una ampliación a causa de la necesidad de agregar caracteres. Ésta es la razón por la que se incluye la tabla ASCII de 8bits, 4 bits de zona y 4 de dígito (tabla 3-3). Los primeros 128 caracteres son los originales. Los segundos 128 son los agregados y aquí se puede hablar de caracteres “no oficiales”, dado que las tablas no guardan uniformidad; en su mayoría, representan caracteres gráficos y no son esenciales. Algunos ejemplos son el símbolo del Yen japonés –D9h–, el franco –F9h–, el signo de interrogación de apertura que utiliza el idioma español –8Ah–, caracteres de contorno (doble línea, línea simple), caracteres de brillo o intensidad que se superponen en una pantalla sobre otros caracteres, etcétera.

ASCII de 8 bits: a causa de la necesidad de agregar caracteres, el código se amplió para poder representar 128 símbolos más.

Tabla 3-3. Tabla de código ASCII ampliado (de 8 bits).

<i>HEX. N°</i>	<i>HEX. N°</i>	0	1	2	3	4	5	6	7
	BITS 7654 3210	0000	0001	0010	0011	0100	0101	0110	0111
0	0000	NUL		SP	0	@	P	'	p
1	0001		DCI	!	1	A	Q	a	q
2	0010		DC2	"	2	B	R	b	r
3	0011		DC3	#	3	C	S	c	s
4	0100		DC4	\$	4	D	T	d	t
5	0101			%	5	E	U	e	u
6	0110			&	6	F	V	f	v
7	0111	BEL		'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF		*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[k	{
C	1100	FF		,	<	L	\	l	:
D	1101	CR		-	=	M]	m	}
E	1110	SO		.	>	N	^	n	~
F	1111	SI		/	?	O	-	o	DEL

<i>HEX. N°</i>	8	9	A	B	C	D	E	F	
	BITS 7654 3210	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	ç	É	á	í	À	Đ	a	=
1	0001	ü	æ	í	í	Á	Ñ	ß	±
2	0010	é	Æ	ó	í	Â	Ó	G	=
3	0011	â	ô	ú	³	Ã	Ó	p	=
4	0100	ä	ö	ñ	'	Ä	Ô	S	ƒ
5	0101	à	ò	Ñ	µ	Å	Õ	s)
6	0110	å	û	ª	¶	Æ	Ö		÷
7	0111	ç	ù	º	·	ç	×	t	~
8	1000	ê	ÿ	¿	,	È	Ø	F	º
9	1001	ë	Ö	¬	¹	É	Ù	Θ	·
A	1010	è	Ü	¬	º	Ê	Ú	O	.
B	1011	í	¢	½	»	Ë	¡	d	√
C	1100	î	£	¼	¼	Ì	—	8	n
D	1101	ì	¥	¡	½	Í	¡	f	²
E	1110	Ä	þ	«	¾	Ï	¡	e	¡
F	1111	Å	f	»	¿	Ϊ	—	n	□

3.3.3 Delimitación de strings

Las cadenas de caracteres alfanuméricas (*strings*), procesadas y almacenadas en la computadora, pueden ser:

- De **longitud fija**: cada dato ocupa un número determinado de bytes fijo. Por lo tanto, el comienzo y el final de cada dato puede conocerse con rapidez.
- De **longitud variable**: para determinar el final de cada dato hay dos métodos:
 - Cada dato tiene en su inicio un campo en el que se indica la longitud en bytes, como lo muestra la figura 3.3.

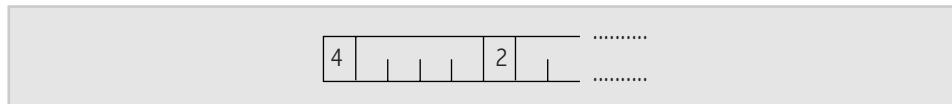


Fig. 3.3. Longitud de bytes.

- Cada dato se separa de los colindantes mediante un símbolo específico o separador, que se utiliza para marcar el final de la cadena de caracteres, pero no se considera parte de ésta. Hay dos separadores muy usados:
 - 1) El “byte 0” o NUL.
 - 2) El byte con un código identificado en la tabla ASCII como de “retorno de carro” (CR), que por lo general se utiliza para el final de una línea de texto, que en el teclado equivale a la tecla <ENTER>. En la figura 3.4 se observa el caso que se menciona.



Fig. 3.4. CR (carriage return).

Para dar un ejemplo, el *string* “PRIMER AÑO” se representa en ASCII de 8 bits (en hexadecimal) como:

P R I M E R A Ñ O CR
50 52 49 4D 45 52 20 41 A5 4F 0D



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

3.4 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. Son códigos de representación de números y se los denomina códigos ponderados, porque adjudican cierto peso a los 1 binarios, según la posición que ocupan en el bloque, por lo que se debe verificar que la suma de los pesos de cada combinación sea igual al número decimal representado.

Aquí se mencionarán tres clases de códigos BCD, el BCD puro y dos derivaciones de él, cuya importancia radica en su capacidad para ser códigos autocomplementarios; ellos son el BCD exceso tres y el BCD 2421.

3.4.1 BCD puro o natural

Se denomina así porque los pesos en cada bloque coinciden con el valor de los 4 primeros pesos del sistema binario puro. Ésta es la razón por la que al BCD puro se lo suele llamar BCD 8421. En la tabla 3-4 se encuentran las equivalencias entre decimal y BCD puro.

Tabla 3-4. BCD puro.

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



Si, por ejemplo, se quiere expresar el número $5_{(10)}$ en BCD puro u 8421, su equivalente es $0101_{(BCD)}$; esto es correcto, ya que si se suman los pesos correspondientes a los 1 contenidos dentro del número 0101 ($4 + 1$) se obtiene el valor 5.

Algunos ejemplos permitirán que se observen las diferencias de representación de un número en los distintos sistemas.

Valor decimal	Binario puro	BCD puro
$15_{(10)}$	$1111_{(2)}$	$0001\ 0101_{(BCD)}$
$256_{(10)}$	$2^8 = 10000000_{(2)}$	$0010\ 0101\ 0110_{(BCD)}$
$1_{(10)}$	$1_{(2)}$	$0001_{(BCD)}$

Nótese que al eliminar los dígitos de zona de “un número” en el código ASCII (zona 3), se obtiene la combinación de bits que representa el dígito BCD correspondiente.

3.4.1.1 BCD empaquetado

La misma ALU que opera en binario puro puede operar en BCD. Lo que cambia es la forma en que se producen las distintas etapas para llevar a cabo la operación. Estas etapas están definidas por instrucciones en el programa, que permiten la gestión de la operación, considerando como base del sistema la decimal y no la binaria.

Los dispositivos electrónicos capaces de realizar la operatoria aritmética permiten almacenar los datos decimales en una forma codificada en sistema binario. Como ya se indicó, los datos del exterior suelen introducirse en la computadora en códigos alfanuméricicos con formato de caracteres. En ASCII los números se representan con octetos divididos en dos partes: zona y dígito. Los números así representados se denominan “**zoneados**” (*unpacked*). Si estos datos numéricos se operaran en la ALU como fueron ingresados, los bits de zona ocuparían, en principio, un lugar innecesario y provocarían resultados incorrectos al operarlos aritméticamente, lo que complicaría el cálculo.

En la tabla de código alfanumérico la zona distingue, por ejemplo, números de letras. A una cadena de caracteres que va a tratarse como un número en la ALU “le sobra la zona”, por lo tanto, una secuencia de instrucciones en el programa la elimina.

El producto final al eliminar la zona y agrupar los dígitos en forma reducida se conoce como “empaquetado” y cada dígito queda representado empleando sólo 4 bits de “dígito”, definido en el byte que representa el carácter numérico.

Un grupo de dígitos BCD constituye un número u operando empaquetado; el signo se incluye en la cabecera o la cola del número, por lo general, en la parte sobrante del octeto que representa el último octeto. Los operandos empaquetados ocupan un número entero de octetos y, por lo tanto, un operando signado, con cantidad par de dígitos decimales, deberá llenarse con un cero en la posición más significativa.

Este tipo de convenio permite el tratamiento aritmético considerando la base decimal, y si bien permite una conversión simple y rápida (eliminar la zona y considerar el signo), la capacidad de la ALU admite que sólo un grupo reducido de dígitos decimales pueda operarse en forma aritmética cada vez; por lo tanto, el tratamiento de dos operandos largos requiere varios accesos a la ALU (hipotéticamente, sumar con una calculadora de sólo dos dígitos dos operandos de 10, implica 5 sumas parciales con el control correspondiente de acarreos).

Un número desempaquetado ocupa un byte, en el que los bits de orden superior son los correspondientes a la zona del código alfanumérico, y se utiliza para el ingreso o para que pueda "mostrarse" al usuario.

+3456	desempaquetado: 2B 33 34 35 36
	empaquetado: 03456B _____ Signo + (B)
+279	desempaquetado: 2B 32 37 39
	empaquetado: 279B
-3456	desempaquetado: 2D 33 34 35 36
	empaquetado: 03456D _____ Signo - (D)
-279	desempaquetado: 2D F2 F7 F9
	empaquetado: 279D

Las operaciones "empaquetar" y "desempaquetar" se indican en forma explícita en el programa con instrucciones que las definan.

3.4.2 BCD exceso tres

Se obtiene a partir del BCD puro, sumando un 3 binario a cada cifra decimal. Así se logra un código simétrico o autocomplementario, que facilita la operación de hallar el complemento de un número (complemento restringido decimal, complemento lógico o negación) sólo con invertir sus dígitos.

En la figura 3.5 se muestran las equivalencias entre decimal y BCD exceso tres. Obsérvese que 0111 (4) es el complemento restringido de 1000 (5), o que 0011 (0) es el complemento restringido de 1100 (9).

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

Fig. 3.5. Código BCD exceso tres.

3.4.3 BCD AIKEN o 2421

Este código adjudica a cada dígito "1" binario el peso que resulta de la combinación 2421, en lugar del BCD puro que impone un peso igual a 8421. Por ejemplo, el número 7 en BCD 2421 es "1101" y surge de la suma de los dígitos "1" multiplicados por sus pesos, o sea, $2 + 4 + 1 = 7$.

También se usa en gran medida por su simetría, que permite hallar con facilidad el complemento de un número, o sea que cuando una operación BCD involucra el complemento restringido de un dígito, éste se obtiene invirtiendo los bits de la combinación de la misma forma que se hace en sistema binario. En la figura 3.6 se observan las equivalencias entre decimal y BCD 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

Código autocomplementario

Fig. 3-6. Código BCD Aiken.

Continuando con los ejemplos presentados, se pueden observar las diferencias de representación de un número en estos últimos dos sistemas.

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

3.5 Códigos de representación numérica no decimal

Estos códigos permiten la operación aritmética de datos en sistema binario o en otras bases no decimales. Los operandos ingresan primero en código alfanumérico para luego convertirse a alguno de estos convenios. Si bien la conversión es más compleja que cuando se opera en sistema decimal, los datos que intervendrán en cálculos reducen su tamaño en grado considerable, con lo que se operan con mayor rapidez. A modo de ejemplo, se puede recordar cuántos dígitos ocupa el número $15_{(10)}$ en un convenio decimal, en contraposición al binario:

$$0001\ 0101_{(BCD)} \quad 1111_{(2)}$$

3.5.1 Coma o punto fijo sin signo (enteros positivos)

Es el formato más simple para representar números enteros positivos, expresados como una secuencia de dígitos binarios:

$$X_{n-1}, X_{n-2}, \dots, X_2, X_1, X_0$$

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

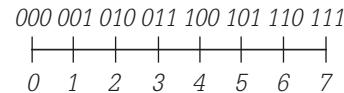


El rango de variabilidad para un número x en este formato es:

$$0 \leq x \leq 2^n - 1$$

donde "n" es la cantidad de dígitos que componen el número; también se la conoce como tipo de dato "ordinal sin signo".

Por ejemplo, para un formato de $n = 3$ bits, el rango de variabilidad estará comprendido entre 0 y 7. Expresado de otra manera, permite representar los números comprendidos en ese rango, como se observa en la recta de representación siguiente:



Por lo tanto, si se suman dos números cuyo resultado supere el rango de variabilidad establecido, este resultado será erróneo, ya que perderá el dígito más significativo. El ejemplo siguiente muestra esta condición para $n = 3$:

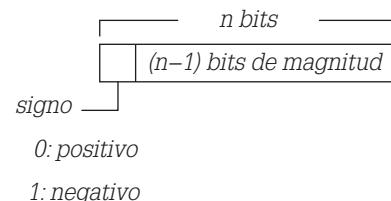
Comprobación

$$\begin{array}{r}
 101 \\
 + \underline{111} \\
 \hline
 1\underline{100}_{(2)} \\
 \end{array}
 \qquad
 \begin{array}{r}
 5 \\
 + \underline{7} \\
 \hline
 12_{(10)} \\
 \end{array}$$

Las unidades de cálculo en una CPU actualizan “señalizadores”, llamados *flags* o banderas, que indican, entre otras, esta circunstancia de desborde u *overflow* que, para este ejemplo en particular, implica la pérdida del bit de orden superior.

3.5.2 Coma o punto fijo con signo (enteros)

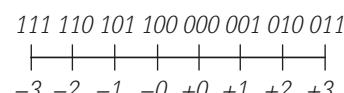
Es igual al formato anterior, pero reserva el bit de extrema izquierda para el signo, de manera que si ese bit es igual a 0 indicará que el número es positivo; asimismo, si, por el contrario, es igual a 1 indicará que el número es negativo. A este tipo de representación también se la llama “magnitud con signo”.



El rango de variabilidad para los binarios puros con signo es:

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

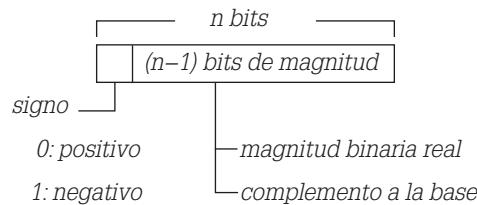
Para el ejemplo de $n = 3$ bits, el rango de variabilidad estará comprendido entre -3 y $+3$, según la recta de representación siguiente:



Nótese la presencia de un cero positivo y otro negativo. Este último surge del cambio de signo, 0 por 1, para esa misma magnitud.

3.5.3 Coma o punto fijo con signo con negativos complementados a “2” (enteros)

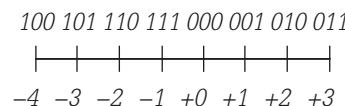
Es igual al formato anterior, reservando el bit de extrema izquierda para el signo, de manera que si ese bit es igual a 0, indicará que el número es positivo; por el contrario, si ese bit es igual a 1, indicará que el número es negativo. La diferencia radica en que los $n - 1$ bits restantes se representan con su magnitud binaria real para el caso de los positivos y en el complemento a la base de la magnitud para el caso de los negativos.



El rango de variabilidad para los binarios con signo en complemento a la base es:

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

Para $n = 3$ bits, el rango de variabilidad estará comprendido entre -4 y $+3$, como se muestra en la recta de representación siguiente:



La representación de los números negativos surge de aplicar la definición para hallar el complemento a la base de un número; por lo tanto, el complemento del número $+2$ con un formato de 3 bits es:

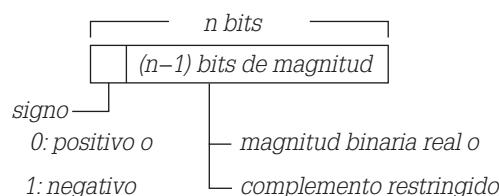
$$\begin{array}{r} 1000 \\ - 010 \\ \hline 110 \end{array}$$

De forma similar, el complemento del número $+3$ es:

$$\begin{array}{r} 1000 \\ + 011 \\ \hline 101 \end{array}$$

3.5.4 Coma o punto fijo con signo con negativos complementados a “1” (enteros)

Es igual al formato anterior, reservando el bit de extrema izquierda para el signo, de manera que si ese bit es igual a 0 , indicará que el número es positivo; por el contrario, si ese bit es igual a 1 , indicará que el número es negativo. No obstante, los $n-1$ bits restantes se representan con su magnitud binaria real para el caso de los positivos y en complemento restringido de la magnitud para el caso de los negativos.

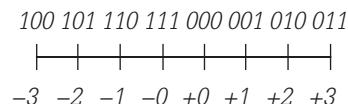


El rango de variabilidad para los binarios puros con signo en complemento restringido o complemento a 1, es:

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

En esta codificación también hay dos “ceros”, uno positivo y otro negativo.

Luego, para $n = 3$ la recta representativa del rango de variabilidad será:



3.5.5 Reales en coma o punto flotante (números muy grandes y números reales)

Los números analizados hasta ahora se pueden clasificar como enteros sin signo y enteros signados, en general limitados por un formato de 8, 16, 32 o 64 bits; sin embargo, a veces es necesario operar datos mayores (por ejemplo, para programas que resuelvan cálculos científicos). La primera solución para tratar números muy grandes es ampliar el formato a tres, cuatro u ocho octetos y obtener así valores decimales que oscilan entre alrededor de los 8 millones y los 9 trillones (considerando enteros signados). De todas maneras, en los cálculos no siempre intervienen valores enteros y es entonces donde aparece el inconveniente de cómo representar el punto para manejar cantidades fraccionarias. Por un lado, el punto debería expresarse con un dígito "0", o bien, con un dígito "1", dentro de la cadena de bits que representa al número binario; por otro lado, el punto no asume una posición fija sino variable dentro de la cadena, o sea que si el punto se representa con un bit "1" no se podrá diferenciar en una secuencia "1111" el binario 1111 del binario 111,1, o bien del binario 1,111. La solución para este problema es eliminar el punto de la secuencia e indicar de alguna manera el lugar que ocupa dentro de ella. De esta forma, los números se representan considerando dos partes. La primera indica los bits que representan al número y se llama **mantisa**, la otra indica la posición del punto dentro del número y se llama **exponente**. Esta manera de tratar números fraccionarios tiene cierta semejanza con lo que se conoce en matemática como notación científica o exponential. Veamos cómo se puede representar un número decimal:

$$3,5_{(10)} = \begin{cases} 35 \cdot 10^{-1} & \text{o bien, es igual a} \\ 0,35 \cdot 10^{+1} & \end{cases}$$

En el primer caso, 35 es un número entero que, multiplicado por la base 10 y elevado a la -1 , expresa el mismo número inicial sin representar la coma dentro de él. Así, "35" es la mantisa y " -1 ", el exponente, que se puede leer como "el punto está **un** lugar a la **izquierda** del dígito menos significativo de la mantisa".

En el segundo caso, 0,35 es un número fraccionario puro (se indica entonces que es una fracción **normalizada**, dado que se supone que el punto está a la izquierda del dígito más significativo y distinto de cero de la mantisa). Luego, 0,35 multiplicado por la base 10 y elevado a la $+1$ expresa el mismo número inicial, sin representar la coma dentro de él. En este caso, " $+1$ " indica que el punto está **un** lugar a la **derecha** del dígito más significativo de la mantisa.

Cuando se trata de números binarios con punto fraccionario se utilizan, entonces, dos entidades numéricas, una para la mantisa (por lo general una fracción normalizada y con signo) y otra para el exponente. La base del sistema es siempre conocida, por lo tanto, no aparece representada. Este nuevo formato de números se conoce como binarios de punto (coma) flotante, ya que el punto varía su posición ("flota") según el contenido binario de la entidad 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 dere-

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

El exponente representa la posición de la coma o punto decimal en la mantisa.

cha o izquierda respecto del **origen**. En la memoria sólo se almacenarán la mantisa "M" y la potencia "P" en binario. 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. Si se considera el registro dividido en partes iguales, 8 bits para la mantisa entera y 8 bits para el exponente, el entero mayor se calcula según la fórmula siguiente:

$$+(2^7 - 1) \cdot 2^{+(2^7 - 1)} = +127 \cdot 2^{+127}$$

que implica que al binario representativo del entero 127 se le agregan 127 ceros a derecha; de modo que enteros muy grandes encuentran su representación en esta modalidad.

Si ahora se representa el valor $3,5_{(10)}$ en binario 11,1 resulta:

$+3,5_{(10)} = +11,1_{(2)}$ expresado en notación científica es igual a

$$+111,0_{(2)} \cdot 10_{(2)}^{-1(2)} = +7_{(10)} \cdot 2_{(10)}^{-1(10)} = +7/2 = +3,5_{(10)} \text{ mantisa entera}$$

$$+0,111_{(2)} \cdot 10_{(2)}^{+10(2)} = +0,875_{(10)} \cdot 2_{(10)}^{+2(10)} = +3,5_{(10)} \text{ mantisa fraccionaria}$$

Dado que los registros que operen estos datos tendrán una longitud fija, se asume un formato de "m" bits para la mantisa y "p" bits para el exponente, entonces se puede definir el rango de representación de números reales, que es siempre finito. O sea que determinados reales son "representables" en el formato y otros quedan "afuera" de él.

Cuando el resultado de una operación supera los límites mayores definidos por el rango, entonces es incorrecto. El 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*.

Cuando el resultado de una operación cae en el hueco definido por los límites inferiores del rango (del que se excluye el cero), esto es

$$0 > x > 0,1 \quad \text{o} \quad 0 < x < 0,1$$

es incorrecto. El error se conoce como *underflow* de resultado.

3.5.5.1 Convenios de representación en punto flotante con exponente en exceso

Para eliminar el signo del exponente se utilizan convenios que agregan al exponente un exceso igual a 2^{p-1} . Esta entidad nueva se denomina característica y es igual a:

CARACTERÍSTICA =	$\pm P$	+	2^{p-1}
EXPONENTE			EXCESO

Siguiendo con el ejemplo de $+3,5_{(10)}$, pero en este caso en un formato de m = 8 (cantidad de bits de la mantisa, incluido el signo) y p = 8 (cantidad de bits de la potencia), el exceso será:

$$2^{p-1} = 2^7 = 128 \quad (10000000_2) \quad \text{donde "p" es la cantidad de bits de la característica.}$$

En este convenio la mantisa se representa en la forma de signo y magnitud, o sea, **no se utiliza el complemento para los negativos**. Ahora bien, la forma en que la computadora "ve" la mantisa puede ser entera o fraccionaria y sin normalizar o normalizada. Analicemos las diferentes formas que puede adoptar la mantisa y su manera de representación, siempre para el mismo ejemplo.



Overflow: es un error que se produce cuando el resultado se encuentra fuera de los límites superiores del rango.

Para una mantisa **entera** será:

Formato alojado en una locación de memoria

$$3,5_{(10)} = 11,1_{(2)} = 111 \cdot 10^{-1}$$

Característica = $128 - 1 = 127$

exc. exp.

CARÁCTER	S	MANTISA
01111111	0	0000111

+ (positivo)

Si la mantisa fuese **fraccionaria** la forma de representarla sería:

$$3,5_{(10)} = 11,1_{(2)} = ,0000111 \cdot 10^{+10}$$

Característica = $128 + 6 = 134$

CARÁCTER	S	MANTISA
10000110	0	.0000111

En general, la manera de representar la mantisa es normalizada; por lo tanto, si se necesita expresar el número anterior con mantisa **fraccionaria y normalizada**, éste será:

$$3,5_{(10)} = 11,1_{(2)} = ,111 \cdot 10^{+10}$$

Característica = $128 + 2 = 130$

CARÁCTER	S	MANTISA
10000010	0	.1110000

En la tabla 3-5 se muestran las correspondencias entre el exponente y la característica.

Tabla 3-5. Correspondencias entre el exponente y la característica.

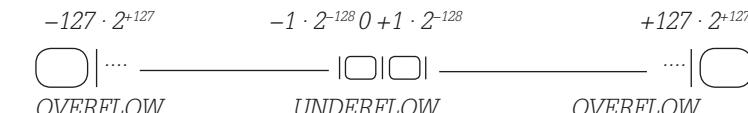
Si el exponente es	la característica será
-128	0
-1	127
0	128
127	255

Analicemos la tabla 3-6, en la que se observan los valores máximos y mínimos que se pueden representar con $m = 8$ y $p = 8$, con una **mantisa entera sin normalizar** (excedida en 128).

Tabla 3-6. Valores máximos y mínimos representados con $m=8$ y $p=8$

	Mantisa	Carácter	Mant.	Poten.	Resulta de
- el mayor nro. positivo es:	0 1111111	1111111	+127	+127	$255 - 128 = +127$
- el menor nro. positivo es:	0 0000001	0000000	+1	-128	$0 - 128 = -128$
- el menor nro. negativo es:	1 0000001	0000000	-1	-128	$0 - 128 = -128$
- el mayor nro. negativo es:	1 1111111	1111111	-127	+127	$255 - 128 = +127$

Los valores máximos y mínimos de la tabla se ven reflejados en la recta de representación siguiente:



Genéricamente:

- El número mayor positivo queda definido como $+(2^{m-1} - 1) \cdot 2^{2(p-1)-1}$
- El número mayor negativo queda definido como $-(2^{m-1} - 1) \cdot 2^{2(p-1)-1}$
- El número menor positivo queda definido como $+2^{-2^{(p-1)}}$
- El número menor negativo queda definido como $-2^{-2^{(p-1)}}$

En resumen, los valores máximos y mínimos que se pueden representar con una mantisa entera sin normalizar son los que se visualizan en la tabla 3-7.

Tabla 3-7. Límites de overflow y underflow.

$$\pm(2^{m-1} - 1) \cdot 2^{2^{(p-1)}-1} \quad \pm1 \cdot 2^{-[2^{(p-1)}]}$$

En los rangos de variabilidad la recta no es siempre continua, porque a medida que el exponente "flota" fuera del límite del registro se "agregan ceros" a derecha o a izquierda de la mantisa y, por lo tanto, se expresan valores enteros o fracciones puras muy pequeñas, por lo que quedan sin representación los valores intermedios.

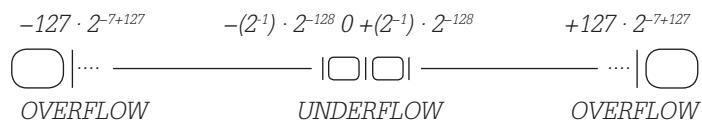
Con una **mantisa normalizada fraccionaria** donde $m = 8$ y $p = 8$ (excedida en 128), los valores máximos y mínimos que se pueden llegar a representar son los que se muestran en la tabla 3-8. Recuerde que 127 es el valor correspondiente a una mantisa entera, entonces, para calcular el valor 0,1111111 se lo debe multiplicar por 2^{-7} , lo que provoca el desplazamiento del punto de extrema derecha a extrema izquierda:

$$+127 \cdot 2^{-7} = \frac{+127}{+128} = 0.99218..$$

Tabla 3-8. Valores máximos y mínimos.

	Mantisa	Carácter	Mant.	Poten.
- el mayor nro. positivo es:	0 1111111	11111111	$127 \cdot 2^{-7}$	+127
- el menor nro. positivo es:	0 1000000	00000000	+.1	-128
- el menor nro. negativo es:	1 1000000	00000000	-.1	-128
- el mayor nro. negativo es:	1 1111111	11111111	$-127 \cdot 2^{-7}$	+127

Los valores máximos y mínimos de la tabla, para mantisa fraccionaria normalizada, se reflejan en la recta de representación siguiente:



Por último, los valores máximos y mínimos que se pueden representar con una mantisa fraccionaria normalizada son los que se visualizan en la tabla 3-9.

Tabla 3-9. Límites de overflow y underflow.

$$\pm(2^{m-1} - 1) \cdot 2^{-(m-1)} \cdot 2^{2^{(p-1)}-1} \quad \pm(2^{-1}) \cdot 2^{-[2^{(p-1)}]}$$

Los sistemas de representación de datos en punto flotante analizados hasta ahora nos permiten introducirnos en el estudio de otros, cuya comprensión hubiese resultado confusa sin este primer acercamiento.

3.5.5.2. Punto flotante "exceso 127"

En esta convención el número se representa en formato de 4 octetos:

1 bit 8 bits $m = 23$ bits mantisa

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

1, (posición supuesta del bit implícito)

El cálculo de la característica o el exponente responde al criterio empleado en la convención anterior:

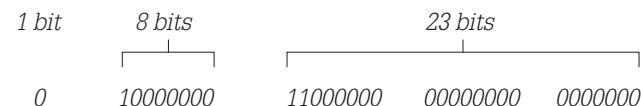
$$\pm P + (2^{g-1} - 1) = \pm P + (2^g - 1) = \pm P + 127_{(10)}$$

El primer bit corresponde al signo de la mantisa (0 positiva, 1 negativa) y en los 31 bits restantes se representa la característica y su magnitud binaria real. El lugar “supuesto” dado que no se representa para su almacenamiento, es la posición entre la característica y los bits de mantisa. Como la mantisa es una fracción pura normalizada, el primer bit luego de la coma debe ser 1. Por esta razón este “1” tampoco se almacena y “se supone” antes de la coma. Por lo tanto, la mantisa está constituida por los bits a la derecha de ella hasta llegar a 23.

1, $b_{22}b_{21}b_{20}\dots\dots\dots b_3b_2b_1b_0$ 23 bits de mantisa

Si ahora representamos $+3,5_{(10)} = +11,1_{(2)} = +1,11_{(2)} \cdot 10^{+1}_{(2)}$ calculamos la *característica* como $(+1) + 0111111 = 10000000$

El signo es positivo, por lo tanto, el formato queda:



El seguimiento de otros ejemplos facilitará su entendimiento:

$$+4,0_{(10)} = +100_{(2)} = 1,0_{(2)} \cdot 10^{+10}_{(2)}$$

0 10000000 00000000 00000000 00000000

El exponente $+10_{(2)}$ significa “desplazar la coma a la derecha dos lugares”.

$$+16,0_{(10)} = +10000_{(2)} = 1,0_{(2)} \cdot 10^{+100}_{(2)}$$

0 10000011 00000000 00000000 00000000

El exponente $+100$ significa “desplazar la coma a la derecha cuatro lugares”.

$$-1,0_{(10)} = -1_{(2)} = -1,0_{(2)} \cdot 10^0_{(2)}$$

1 01111111 00000000 00000000 00000000

El exponente 0 significa “no desplazar la coma”.

$$-0,5_{(10)} = -0,1_{(2)} = -0,1_{(2)} \cdot 10^{-1}_{(2)}$$

1 01111110 00000000 00000000 00000000

El exponente -1 significa “desplazar la coma a la izquierda un lugar”.

$$+0,125_{(10)} = +0,001_{(2)} = 1,0_{(2)} \cdot 10^{-1}_{(2)}$$

El exponente -11 significa “desplazar la coma a la izquierda tres lugares”.

Estos dos últimos ejemplos representan casos de números menores que cero, por lo tanto, la coma sufre un “desplazamiento a izquierda”, que se representa en convenio con el signo del exponente negativo y la característica menor que 127.

De la observación de los últimos cinco ejemplos se pueden sacar las conclusiones siguientes:

- Los números negativos sólo se diferencian de los positivos por el valor del bit de signo asociado a la mantisa.
- Toda potencia de 2 se representa con mantisa $1,00\ldots_{(2)}$ y toma su valor según el desplazamiento del punto.
- Con una misma mantisa y distintas características se representan diferentes valores decimales.
- El punto puede “flotar” aun fuera del límite físico que impone el formato.

Pasos para la representación de un número en este formato

1. Convertir el número decimal a binario.
2. Normalizar, eliminar el bit denominado “implícito” y calcular el valor de “P” para determinar el desplazamiento del punto.
3. Calcular y representar la característica.
4. Acomodar el signo y la mantisa, considerando sólo los dígitos fraccionarios hasta completar el formato.

Ahora considérese el ejemplo siguiente:

$$-0,2_{(10)} = -0,\overline{0011}_{(2)} = -0,1100110011\ldots_{(2)} \cdot 10_{(2)}^{-11_{(2)}} = -1,\overline{1001} \cdot 10_{(2)}^{-11_{(2)}}$$

Esta conversión no es exacta, o sea que la fracción binaria hallada es periódica y, por lo tanto, se aproxima a la fracción decimal original.

El error varía en relación con la cantidad de períodos que se consideren. Cuando se normaliza, los ceros a derecha del punto desaparecen, se indica su existencia con el valor del exponente $-11_{(2)}$ (tres lugares a izquierda del origen de la coma). El número representado queda así:

1 01111100 10011001 10011001 1001100 truncated

110011001...

En la mantisa se repite el período tantas veces como el formato lo permite hasta que se trunca.

Con este último ejemplo se pretende que quede claro que si la cantidad de bits de la mantisa aumenta, mejor será la precisión del número.

A continuación, se presenta un ejemplo que servirá de ejercitación. Representemos en punto flotante convenio exceso -127 los números $-5001,25_{(10)}$ y $+10,1_{(10)}$.

$$-5001,25_{(10)} = -1001110001001,01 = -1,00111000100101_{(2)} \cdot 10_{(2)}^{+1100}$$

1 11110011 00111000100101000000000

-	115	bit implícito
---	-----	---------------

Precisión simple y precisión doble

En esta convención la cantidad de bits que representan el número admite dos variantes:

- Formato de precisión simple: cuatro octetos.
- Formato de precisión doble: ocho octetos.

Los ejemplos aportados pertenecen al formato de precisión simple. La determinación de los límites sobre la recta real nos permite observar la mejora en la precisión del formato mayor. Queda a criterio del programador cuándo utilizar una u otra, al momento de definir en sus programas variables reales; se debe tener en cuenta que una mejora en la precisión implica el almacenamiento de mayor cantidad de bits, y esto tendrá desventajas que deberán evaluarse.

3.6 Representaciones redundantes

3.6.1 Códigos de detección y/o corrección de errores. Introducción

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. De todas formas, la cantidad de errores que puedan producirse son mensurables probabilísticamente. 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.

3.6.2 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	<i>se agrega un uno que permita obtener paridad par</i>
01001011 0	<i>se agrega un cero para lograr paridad par</i>

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.

3.6.3 Paridad horizontal a nivel de bloque

Por cada bloque de caracteres se crea un byte con bits de paridad. El bit 0 será el bit de paridad de los bits 0 del bloque; el bit 1, el de paridad de los bits 1 del bloque, y así sucesivamente.

3.6.4 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.

Bit de paridad: es un bit redundante agregado a una cadena de bits en la que se pretende detectar un posible error.

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
$PV - 0 1 1 1 1$	$PV - 0 1 1 1 1$
<i>sin error</i>	<i>con error</i>

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. En la práctica este código no se considera óptimo, debido a la cantidad de bits de paridad que se deben agregar, por lo que suele utilizarse el código de Hamming.

3.6.5 Código de Hamming

Este código permite detectar y corregir los errores producidos en una transmisión con sólo agregar p bits de paridad, de forma que se cumpla la relación siguiente:

$$2^p \geq i + p + 1$$

donde i es la cantidad de dígitos binarios que se han de transmitir y p es la cantidad de bits de paridad.

Supongamos que se desean transmitir 4 dígitos binarios. Según el método de Hamming a éstos deberá agregarse la cantidad de bits de paridad necesarios para satisfacer la relación enunciada antes:

$$i = 4 \quad \text{para que la relación } 2^p \geq i + p + 1 \text{ se cumpla debe ser}$$

$$p = ? \quad p = 3, \text{ luego} \quad 2^3 \geq 4 + 3 + 1$$

De esta relación se deduce que la transmisión será de 4 bits de información más 3 de paridad par. Se debe tener en cuenta que con 2 bits de paridad se puede corroborar la transmisión de 1 dígito de información, con 3 de paridad se corroboran hasta 8 bits de información, con 4 de paridad se corroboran hasta 11 de información, y así sucesivamente.

En la cadena de 7 bits del ejemplo: b_1, b_2, \dots, b_7 , son bits de paridad los que ocupan las posiciones equivalentes a las potencias de dos y el resto son bits de datos.

La tabla siguiente es la guía para determinar la distribución y el cálculo de los bits de paridad y su corroboración luego de la transmisión de hasta 11 dígitos binarios.

$p_1 = b_1$	$p_2 = b_2$	$p = b_4$	$p_4 = b_8$
$p_1 = b_1$	*		
$p_2 = b_2$		*	
$i_1 = b_3$	*	*	
$p_3 = b_4$			*
$i_2 = b_5$		*	*
$i_3 = b_6$		*	*
$i_4 = b_7$	*	*	*
$p_4 = b_8$			*
$i_5 = b_9$		*	*
$i_6 = b_{10}$		*	*
$i_7 = b_{11}$	*	*	*
$i_8 = b_{12}$		*	*
$i_9 = b_{13}$	*	*	*
$i_{10} = b_{14}$		*	*
$i_{11} = b_{15}$	*	*	*

Por lo tanto, para corroborar la transmisión de cuatro bits de información, deberán verificarse las igualdades siguientes:

$$\begin{aligned} p_1 &= b_1 + b_3 + b_5 + b_7 \\ p_2 &= b_2 + b_3 + b_6 + b_7 \\ p_3 &= b_4 + b_5 + b_6 + b_7 \end{aligned}$$

Si llegara a producirse algún error en la transmisión, (por ejemplo, se transmite el número binario 1101 y se recibe el 1100) el método de Hamming indicará el número del bit donde se produjo el error, mediante el proceso de verificación que realiza en el punto receptor. Veamos:

$$\begin{array}{ll} \text{Se transmiten:} & i_4 \ i_3 \ i_2 \ p_3 \ i_1 \ p_2 \ p_1 \\ & b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \\ & 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \end{array}$$

$$\text{Se reciben: } 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1$$

$$\begin{array}{ll} \text{Verificación:} & p_3 \ p_2 \ p_1 \\ & 1 \ 1 \ 1 = \text{el bit } 7_{(10)} \text{ es el erróneo.} \end{array}$$

Una vez detectado el error la corrección es sencilla, basta con invertir el bit erróneo.

3.7 Resumen

Como vimos, toda información interna se representa en forma binaria. En este capítulo se describieron en particular algunas de las representaciones de datos; veamos, entonces, algunos ejemplos de su utilización desde la perspectiva del lenguaje de programación y el diseño de hardware, considerando a los fabricantes de software y hardware más populares para ilustrar lo expuesto hasta ahora.

En el caso de números de coma flotante, Microsoft utiliza las versiones de *IEEE 754* en el formato de 4 bytes, 8 bytes y 10 bytes, definidas en lenguaje ensamblador con las directivas "DD", "DQ" y "DT", respectivamente. En relación con el hardware, podemos dar como ejemplo su origen en el coprocesador 8087 (ya fuera de uso) para los primeros procesadores Intel de la industria 80x86 y en todas las unidades de coma flotante incluidas en los procesadores actuales del mismo fabricante. Respecto de los lenguajes de programación, para dar un ejemplo de identificación de los tipos de datos podemos utilizar el "C", en el que una variable de coma flotante de 4 bytes se declara como *float*, la variable coma flotante de 8 bytes se declara como *double* y la variable de coma flotante de 10 bytes, como *long double*.

Algunos lenguajes de programación como C, C++ y Java soportan tipos de datos decimales. Este soporte también se encuentra en los procesadores IBM Power 6 y en el *mainframe* de IBM System Z10.

3.8 Ejercicios propuestos

1) La siguiente es una sentencia en lenguaje de alto nivel, $C = A - B$.

Indicar un código que la represente en el programa fuente y establecer cuántos octetos ocuparía en memoria.

2) Para un formato de 32 bits, indicar los rangos de variabilidad en decimal, para números representados en ASCII sin signo y en binario signado en complemento a 2, e indicar cómo se representa el número $+25_{(10)}$ en los dos sistemas.

3) Dado el número $7910_{(10)}$, expresarlo en:

- a) Decimal codificado en binario.
- b) BCD exceso -3.
- c) BCD exceso -2421.
- d) Binario natural.

4) Si D2 es la representación hexadecimal de un número, exprese cuál es ese número en sistema decimal, si se considera un entero:

- a) No signado.
- b) Signado.

5) Expresar en decimal los límites de representación para un formato de 16 bits, si éste contiene:

- a) Un entero no signado.
- b) Un entero signado.

6) Representar $(-18)_{(10)}$ y $(-64)_{(10)}$ con un formato de 16 bits, en punto fijo en complemento restringido y en punto fijo en complemento a la base.

7) Determinar el rango de variabilidad para un formato de punto fijo con negativos complementados a dos en:

- a) 6 bits.
- b) 32 bits.
- c) 64 bits.

8) Dados los siguientes números, expresarlos en punto flotante, convención exceso 127.

- a) $+32_{(10)}$
- b) $+3.25_{(10)}$
- c) $(-0.4)_{(10)}$
- d) $(-10.1)_{(10)}$
- e) $+0.5_{(10)}$

9) La siguiente representación hexadecimal determina una cadena binaria: 945A.

- a) Indicar el valor decimal que representa si la cadena se considera binario natural.
- b) Indicar qué valor decimal representa si la cadena se considera un binario signado en sus tres formas de representación.

- c) Suponiendo que la cadena es la representación de un número en punto flotante en exceso con mantisa entera, donde $m = 8$ y $p = 8$, indicar:
- ¿Cuál es el exceso utilizado?
 - ¿Cuál es el número decimal que representa en la forma $\pm M \cdot B^{\pm p}$?
- 10) Para un formato que permita una mantisa fraccionaria con $m = 8$ y $p = 8$:
- a) Determine los límites de representación, a partir de los cuales se produce *overflow* y *underflow* de resultado.
 - b) ¿Cuál es el exceso que permite emplear este formato?
 - c) ¿Qué permite la utilización de una mantisa con mayor cantidad de bits?
 - d) ¿Cuál sería la consecuencia si "p" tuviera mayor cantidad de bits?
- 11) Determine los límites, a partir de los cuales se produciría *overflow* y *underflow* de resultado, en precisión simple y en precisión doble, para el formato conocido como "exceso-127".
- 12) Suponga que recibe el número $87_{(10)}$ en el buffer de la impresora para ser listado en representación ASCII y que la transmisión utiliza como método de control el código de Hamming:
- a) Desarrolle las funciones para determinar el cálculo de los bits de paridad,
 - b) Suponga que se producen errores en la transmisión en el bit número:
 - 10 para el dígito $8_{(10)}$
 - 13 para el dígito $7_{(10)}$.
 - c) Exprese cómo se desarrolla la detección y la corrección de ambos errores.

3.9 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.



Resumen gráfico del capítulo

Simulación

Permite ingresar un texto y lo codifica en ASCII.

Autoevaluación

Video explicativo (02:44 minutos aprox.)

Audio explicativo (02:44 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*