CAPÍTULO 3

CODIFICACIÓN INTERNA DE LA INFORMACIÓN

El objetivo de este capítulo es mostrar algunas formas habituales en que un ordenador codifica la información. Es conveniente conocer esta codificación: cómo fluye la información de los periféricos hacia el procesador o al revés; y cómo codifica el procesador la información en sus registros, o en la memoria principal, o en los circuitos de la ALU. Y resulta además de utilidad en las tareas del programador, que puede obtener muchas ventajas en el proceso de la información si conoce el modo en que esa información se encuentra disponible en las entrañas del ordenador.

Introducción.

La información, en un ordenador, se almacena mediante datos codificados con ceros y unos. Ya lo hemos visto en los capítulos

precedentes. Ahora, en este capítulo, queremos ver cómo son esos códigos de ceros y unos. No nos vamos a entretener en la codificación de todos los posibles tipos de dato. Hemos centrado principalmente la presentación de este capítulo en el modo cómo se codifican los enteros. Y eso por dos motivos: porque es un código muy sencillo; y porque resulta de gran utilidad conocer esa codificación: el lenguaje C o el Java ofrece herramientas para poder manipular ese código y obtener, si se sabe, resultados interesantes y ventajosos.

Al tratar de los datos a codificar, deberemos distinguir entre la codificación que se emplea para la entrada y salida de datos, y la que el ordenador usa para su almacenamiento en memoria. Por ejemplo, si el usuario desea introducir el valor 412, deberá pulsar primero el cuatro, posteriormente la tecla del uno, y finalmente la del dos. El modo en que el teclado codifica e informa a la CPU de la introducción de cada uno de estos tres caracteres será diferente al modo en que finalmente el ordenador guardará en memoria el valor numérico 412.

Así las cosas, el modo en que un usuario puede suministrar información por teclado a la máquina es mediante caracteres, uno detrás de otro. Estos caracteres podemos clasificarlos en distintos grupos:

1. De Texto:
    a. Alfanuméricos:
        i. **Alfabéticos**: de la ‘a’ a la ‘z’ y de la ‘A’ a la ‘Z’.
        ii. **Numéricos**: del ‘0’ al ‘9’.
    b. **Especiales**: por ejemplo, ‘(‘, ‘)’. ‘+’. ‘?’. ‘@’,…
2. **De control**: por ejemplo, fin de línea, tabulador, avance de página, etc.
3. **Gráficos**: por ejemplo, ‘☺’, ‘♥’, ‘♣’,…

Como el ordenador sólo dispone, para codificar la información, de ceros y de unos, deberemos establecer una correspondencia definida entre el

conjunto de todos los caracteres y un conjunto formado por todas las posibles secuencias de ceros y de unos de una determinada longitud. A esa correspondencia la llamamos código de Entrada/Salida. Existen muchos distintos **códigos de E/S**, algunos de ellos normalizados y reconocidos en la comunidad internacional. Desde luego, cualquiera de estas codificaciones de E/S son arbitrarias, asignando a cada carácter codificado, una secuencia de bits, sin ninguna lógica intrínseca. Estos códigos requieren de la existencia de tablas de equivalencia uno a uno, entre el carácter codificado y el código asignado para ese carácter.

Y, como acabamos de decir, esta codificación es distinta de la que, una vez introducido el dato, empleará el ordenador para codificar y almacenar en su memoria el valor introducido. Especialmente, si ese valor es un valor numérico. En ese caso especialmente, tiene poco sentido almacenar la información como una cadena de caracteres, todos ellos numéricos, y resulta mucho más conveniente, de cara también a posibles operaciones aritméticas, almacenar ese valor con una codificación numérica binaria. En ese caso, estamos hablando de la **representación o codificación interna** de los números, donde ya no se sigue un criterio arbitrario o aleatorio, sino que se toman en consideración reglas basadas en los sistemas de numeración posicional en base dos.

Códigos de Entrada/Salida.

Ya hemos quedado que esta codificación es arbitraria, asignando a cada carácter del teclado un valor numérico que queda codificado en las entrañas del ordenador mediante una cifra en base binaria de una longitud determinada de bits.

La cantidad de bits necesaria para codificar todos los caracteres dependerá, lógicamente, del número de caracteres que se deseen codificar. Por ejemplo, con un bit, tan solo se pueden codificar dos

caracteres: a uno le correspondería el código 0 y al otro el código 1. No tenemos más valores de código posibles y, por tanto, no podemos codificar un conjunto mayor de caracteres. Con dos bits, podríamos codificar cuatro caracteres; tantos como combinaciones posibles hay con esos dos dígitos binarios: 00, 01, 10 y 11.

En general diremos que con *n* bits seremos capaces de codificar hasta un total de 2^n caracteres. Y, al revés, si necesitamos codificar un conjunto de β caracteres, necesitaremos una secuencia de bits de longitud $n \geq \log_2 \beta$. Habitualmente, para un código de representación de caracteres se tomará el menor n que verifique esta desigualdad.

Una vez decidido el cardinal del conjunto de caracteres que se desea codificar, y tomado por tanto como longitud del código el menor número de bits necesarios para lograr asignar un valor de código a cada carácter, el resto del trabajo de creación del código será asignar a cada carácter codificado un valor numérico binario codificado con tantos ceros o unos como indique la longitud del código; evidentemente, como en cualquier código, deberemos asignar a cada carácter un valor numérico diferente.

Desde luego, se hace necesario lograr universalizar los códigos, y que el mayor números de máquinas y dispositivos trabajen con la misma codificación, para lograr un mínimo de entendimiento entre ellas y entre máquinas y periféricos. Para eso surgen los códigos normalizados de ámbito internacional. De entre los diferentes códigos normalizados válidos, señalamos aquí el **Código ASCII** (American Standard Code for Information Interchange). Es el código más ampliamente utilizado. Está definido para una longitud de código $n = 7$ (es decir, puede codificar hasta 128 caracteres distintos), aunque existe una versión del ASCII de longitud $n = 8$ que dobla el número de caracteres que se pueden codificar (hasta 256) y que ha permitido introducir un gran número de caracteres gráficos.

En el código ASCII el carácter 'A' tiene el valor decimal 65 (en hexadecimal 41), y consecutivamente, hasta el carácter 'Z' (valor decimal 90, en hexadecimal 5A), van ordenados alfabéticamente, todas las letras mayúsculas. El alfabeto en minúsculas comienza un poco más adelante, con el código decimal 97 (61 en hexadecimal) para la 'a' minúscula. Las letras 'ñ' y 'Ñ' tienen su código fuera de esta secuencia ordenada. Esta circunstancia trae no pocos problemas en la programación de aplicaciones de ordenación o de manejo de texto. Los caracteres numéricos comienzan con el valor decimal 48 (30 en hexadecimal) para el carácter '0', y luego, consecutivos, están codificados los restantes nueve guarismos de la base diez.

Representación o Codificación Interna de la Información.

La codificación de Entrada/Salida no es útil para realizar operaciones aritméticas. En ese caso resulta mucho más conveniente que los valores numéricos queden almacenados en su valor codificado en base dos.

Por ejemplo, utilizando el código ASCII, el valor numérico 491 queda codificado como 0110100 0111001 0110001. Ese mismo valor, almacenado con 16 bits, en su valor numérico, toma el código 0000000111101011. No resulta mejor código únicamente porque requiere menos dígitos (se adquiere mayor compactación), sino también porque esa codificación tiene una significación inmediatamente relacionada con el valor codificado. Y porque un valor así codificado es más fácilmente operable desde la ALU que si lo tomamos como una secuencia de caracteres de código arbitrario. Es decir, se logra una mayor adecuación con la aritmética.

Vamos a ver aquí la forma en que un ordenador codifica los valores numéricos enteros, con signo o sin signo. Desde luego, existe también una definición y normativa para la codificación de valores numéricos con

decimales, también llamados de coma flotante (por ejemplo, la normativa IEEE 754), pero no nos vamos a detener en ella.

Enteros sin signo.

Para un entero sin signo, el sistema utilizado es tomar como código su valor en base binaria. Sin más.

Por ejemplo, para codificar el valor numérico $N = 175$ con ocho bits tomamos el código 10101111.

Además de saber cómo se codifica el entero, será necesario conocer el rango de valores codificables. Y eso estará en función del número de bits que se emplearán para la codificación.

Si tomáramos un byte para codificar valores enteros sin signo, entonces podríamos codificar hasta un total de 256 valores. Tal y como se ha definido el código en este epígrafe, es inmediato ver que los valores codificados son los comprendidos entre el 0 (código 00000000) y el 255 (código 11111111), ambos incluidos.

Si tomáramos dos bytes para codificar (16 bits), el rango de valores codificados iría desde el valor 0 hasta el valor 65.535 (en hexadecimal FFFF). Y si tomáramos cuatro bytes (32 bits) el valor máximo posible a codificar sería, en hexadecimal, el FFFFFFFF que, en base 10 es el número 4.294.967.295.

Evidentemente, cuantos más bytes se empleen, mayor cantidad de enteros se podrán codificar y más alto será el valor numérico codificado. En general, el rango de enteros sin signo codificados con n bits será el comprendido entre 0 y $+2^n - 1$.

Enteros con signo.

Hay diversas formas de codificar un valor numérico con signo. Vamos a ver aquí una de ellas, la que se emplea en los PC's de uso habitual.

El modo de codificar enteros con signo es el siguiente:

1. El bit más significativo no es un dígito numérico, sino el signo. Se pone a 0 si el entero codificado es positivo; se pone a 1 si el entero codificado es negativo.
2. Los restantes bits se emplean para la codificación del valor numérico. Si el entero es positivo, se codifica en esos bits ese valor numérico en base binaria. Si el entero es negativo, entonces lo que se codifica en esos bits es el complemento a la base del valor absoluto del valor codificado.

Si queremos saber entonces cómo queda codificado, con un byte, el valor numérico -75, debemos hacer los siguientes cálculos: El bit más significativo será 1, porque el entero a codificar es negativo. El código binario del valor absoluto del número es 1001011 (siete dígitos, que son los que nos quedan disponibles). El complemento a la base menos uno de ese valor es 0110100 (se calcula invirtiendo todos los dígitos: de 0 a 1 y de 1 a 0), y el complemento a la base será entonces 0110101 (recuérdese la igualdad 2.6.). Por tanto la representación interna de ese valor será 10110101, que en base hexadecimal queda B5.

Si hubiésemos codificado el entero con dos bytes entonces el resultado final del código sería FFB5.

El rango de valores codificados cambia con respecto al presentado antes con los enteros sin signo, pero la cantidad de valores codificados es la misma en el caso de enteros con signo y enteros sin signo.

Ahora el rango de valores queda centrado en el cero: toma la mitad de los códigos para valores negativos y la otra mitad para valores positivos más el cero. Por ejemplo, con un byte se codifican los enteros

comprendidos entre -128 y +127. Con dos bytes se pueden codificar los enteros comprendidos entre -32.768 y + 32.767. Y con cuatro bytes el rango de valores codificados va desde el -2.147.483.648 hasta el +2.147.483.647.

En general, el rango de valores codificados con n bits será el comprendido entre -2^{n-1} y $+2^{n-1} - 1$.

Recapitulación.

En este capítulo hemos visto cómo se codifica la información dentro del ordenador. Hemos conocido primero cómo se trasfiere la información desde los periféricos hacia la CPU (código de E/S). Y cómo se codifica la información una vez ésta ya ha sido introducida en el ordenador: cómo se almacena en la memoria principal o en los registros internos del ordenador; especialmente nos hemos centrado en la codificación de los enteros.

En el epígrafe siguiente se muestran muchos valores negativos codificados tal y como los almacena el ordenador. Después de terminar el estudio de este capítulo es conveniente practicar y obtener la codificación interna de diferentes valores tomados de forma aleatoria.

Ejercicios.

El mejor modo para llegar a manejar la técnica de codificación de los enteros es la práctica. Queda recogida, en este último epígrafe, la codificación de diferentes valores numéricos, para que se pueda practicar y verificar los resultados obtenidos. Todos ellos son valores negativos, y todos ellos codificados con dos bytes (16 bits, 4 dígitos hexadecimales).

-128	FF80	-127	FF81	-126	FF82	-125	FF83
-124	FF84	-123	FF85	-122	FF86	-121	FF87
-120	FF88	-119	FF89	-118	FF8A	-117	FF8B
-116	FF8C	-115	FF8D	-114	FF8E	-113	FF8F
-112	FF90	-111	FF91	-110	FF92	-109	FF93
-108	FF94	-107	FF95	-106	FF96	-105	FF97
-104	FF98	-103	FF99	-102	FF9A	-101	FF9B
-100	FF9C	-99	FF9D	-98	FF9E	-97	FF9F
-96	FFA0	-95	FFA1	-94	FFA2	-93	FFA3
-92	FFA4	-91	FFA5	-90	FFA6	-89	FFA7
-88	FFA8	-87	FFA9	-86	FFAA	-85	FFAB
-84	FFAC	-83	FFAD	-82	FFAE	-81	FFAF
-80	FFB0	-79	FFB1	-78	FFB2	-77	FFB3
-76	FFB4	-75	FFB5	-74	FFB6	-73	FFB7
-72	FFB8	-71	FFB9	-70	FFBA	-69	FFBB
-68	FFBC	-67	FFBD	-66	FFBE	-65	FFBF
-64	FFC0	-63	FFC1	-62	FFC2	-61	FFC3
-60	FFC4	-59	FFC5	-58	FFC6	-57	FFC7
-56	FFC8	-55	FFC9	-54	FFCA	-53	FFCB
-52	FFCC	-51	FFCD	-50	FFCE	-49	FFCF
-48	FFD0	-47	FFD1	-46	FFD2	-45	FFD3
-44	FFD4	-43	FFD5	-42	FFD6	-41	FFD7
-40	FFD8	-39	FFD9	-38	FFDA	-37	FFDB
-36	FFDC	-35	FFDD	-34	FFDE	-33	FFDF
-32	FFE0	-31	FFE1	-30	FFE2	-29	FFE3
-28	FFE4	-27	FFE5	-26	FFE6	-25	FFE7
-24	FFE8	-23	FFE9	-22	FFEA	-21	FFEB
-20	FFEC	-19	FFED	-18	FFEE	-17	FFEF
-16	FFF0	-15	FFF1	-14	FFF2	-13	FFF3
-12	FFF4	-11	FFF5	-10	FFF6	-9	FFF7
-8	FFF8	-7	FFF9	-6	FFFA	-5	FFFB
-4	FFFC	-3	FFFD	-2	FFFE	-1	FFFF

A modo de ejemplo, mostramos los pasos a seguir para llegar a la codificación interna, en hexadecimal, a partir del valor entero.

- ***Codificación, con 2 bytes, del valor entero*** $(-47)_{10}$.

El bit más significativo estará a 1, porque el entero es negativo. Los restantes 15 bits codifican el valor absoluto del entero. Su valor en binario (con 15 dígitos binarios) es $(47)_{10} = (000\ 0000\ 0010\ 1111)_2$. Su complemento a la base menos uno: $C_1^{15}(000\ 0000\ 0010\ 1111) = 111\ 1111\ 1101\ 0000$ y su complemento a la base: $C_2^{15}(N) = C_1^{15}(N) + 1 = 111\ 1111\ 1101\ 0001$. El código del entero en la representación interna del ordenador será, al añadirle delante el bit del signo, 1111 1111 1101 0001, que en hexadecimal será FFD1.

- ***Codificación, con 2 bytes, del valor entero*** $(-1)_{10}$

  El bit más significativo estará a 1, porque el entero es negativo. Los restantes 15 bits codifican el valor absoluto del entero. Su valor en binario (con 15 dígitos binarios) es: $(1)_{10} = (000\ 0000\ 0000\ 0001)_2$. Su complemento a la base menos uno: $C_1^{15}(000\ 0000\ 0000\ 0001) = 111\ 1111\ 1111\ 1110$ y su complemento a la base: $C_2^{15}(N) = C_1^{15}(N) + 1 = 111\ 1111\ 1111\ 1111$. El código del entero en la representación interna del ordenador será, en hexadecimal, al añadirle el bit del signo, FFFF.

- ***Otras formulaciones que ayudan a practicar y así entender la codificación interna de los enteros:***

  - Indicar el valor numérico que queda codificado con 89D4 (codificación en 16 bits)
  - Si un entero queda codificado con AB (suponiendo una codificación en 16 bits), indique cómo quedaría codificado su valor cambiado de signo.
  - Supongamos una codificación en 16 bits. Indique cuáles de las siguientes codificaciones de enteros con signo corresponden a enteros positivos, y cuáles a enteros negativos: ABCD, 7FFF, A0, FFF0.
  - Ordene los números codificados en la pregunta anterior de menor a mayor.