Introducción a la Computación 2017

1 de abril de 2017

Índice

Sistemas de Numeración

En este primer tema de la unidad veremos las propiedades de los sistemas de numeración más importantes para el estudio de la arquitectura de computadoras, en especial los sistemas **binario y hexadecimal**.

Un sistema diferente

Todos conocemos el método tradicional de contar con los dedos. Como tenemos cinco dedos en cada mano, podemos contar hasta diez. Pero también podemos utilizar un método diferente del tradicional, que resulta ser muy interesante.

- Con este método, al llegar a 5 con la mano derecha, representamos el 6 **sólo con un dedo de la izquierda**. Los dedos de la mano derecha **vuelven a 0**, y seguimos contando con la derecha.
- Cada vez que se agotan los dedos de la mano derecha levantamos un nuevo dedo de la izquierda, y la derecha vuelve a 0.
- Cada dedo en alto de la mano izquierda significa que **se agotó la secuencia de la mano derecha una vez**.

Preguntas

- ¿Hasta qué número se puede representar en este sistema, sólo con dos manos?
- Si agregamos una tercera mano, de un amigo, ¿hasta qué número llegamos?
- ¿Y cómo se representa el 36? ¿Y el 37?
- Y con cuatro manos, ¿hasta qué número llegamos?
- Y, si el número no se puede representar con dos manos, ¿cómo es el procedimiento para saber qué dedos levantar?

Notemos que este método tiene mayor capacidad que el tradicional, ya que podemos contar hasta diez y todavía nos queda mucho por contar con los dedos de ambas manos.

Sistema posicional

Notemos además que esta ventaja se debe a que el método asigna **valores diferentes** a ambas manos. La derecha vale la cantidad de dedos que muestre, pero la izquierda vale **seis por su cantidad de dedos**. Esto se abrevia diciendo que se trata de un **sistema de numeración posicional**.

Al tratarse de un sistema posicional, podemos representar números relativamente grandes con pocos dígitos. En este sistema, disponemos únicamente de **6 dígitos (0, 1, 2, 3, 4, 5)** porque ésos son los que podemos representar con cada mano, es decir, **en cada posición**. Pero los números representables solamente dependen de cuántas manos (o, mejor dicho, de cuántas **posiciones**) podamos utilizar.

Calculando cada posición

En este sistema, dado un número no negativo que se pueda representar con dos manos, podemos saber qué dedos levantar en cada mano haciendo una sencilla cuenta de división entera (sin decimales): dividimos el número por 6 y tomamos el cociente y el resto. **El cociente es el número de la izquierda, y el resto, el de la derecha**.

Tomemos por ejemplo el número 15. Al dividir 15 por 6, el cociente es 2 y el resto es 3. En este sistema, escribimos el 15 como **dos dedos en la izquierda, y tres dedos en la derecha**, lo que podemos abreviar como **(2,3)** o directamente **23** (que se pronuncia **dos tres** porque **no quiere decir veintitrés**, sino **quince**, sólo que escrito en este sistema). Como el dígito 2 de la izquierda vale por 6, si hacemos la operación de sumar $2 \times 6 + 3$ obtenemos, efectivamente, 15.

Base de un sistema de numeración

La **base** de un sistema es la cantidad de dígitos de que dispone, o sea que el sistema decimal habitual es de base 10, mientras que el de los deditos es de base 6.

Número y numeral

Notemos que un mismo número puede escribirse de muchas maneras: en prácticamente cualquier base que se nos ocurra, sin necesidad de contar con los dedos; y que la forma habitual, en base 10, no es más importante o mejor que las otras (salvo, claro, que ya estamos acostumbrados a ella). Otras culturas han desarrollado otros sistemas de numeración y escriben los números de otra manera.

Esto muestra que hay una **diferencia entre número y numeral**, diferencia que es algo difícil de ver debido a la costumbre de identificar a los números con su representación en decimal.

- El **numeral** es lo que escribimos (15, $15_{(10}$ o $23_{(6)}$).
- El **número** es la cantidad de la cual estamos hablando (la misma en los tres casos).

Indicando la base

Anteriormente escribíamos **15** en el sistema de base 6 como **23**. Sin embargo, necesitamos evitar la confusión entre ambos significados de **23**. Para esto usamos índices subscriptos que indican la base. Así,

- **Quince** es $15_{(10}$ porque está en base diez (la del sistema decimal, habitual), y
- $23_{(10}$ es **veintitrés**,
- pero $23_{(6}$ es **dos tres en base 6**, y por lo tanto vale **quince**.

Como 10 es nuestra base habitual, cuando no usemos índice subscripto estaremos sobreentendiendo que hablamos **en base 10**. Es decir, $15_{(10}$ se puede escribir, simplemente, 15.

Cuando queremos pasar un número escrito en una base a un sistema con otra cantidad de dígitos, el procedimiento de averiguar los dígitos que van en cada posición se llama **conversión de base**. Más adelante veremos procedimientos de conversión de base para cualquier caso que aparezca.

Sistema decimal

Si reflexionamos sobre el sistema decimal, de diez dígitos, encontramos que también forma un sistema posicional, sólo que con 10 dígitos en lugar de los seis del sistema anterior.

Cuando escribimos **15** en el sistema decimal, esta expresión equivale a decir "para saber de qué cantidad estoy hablando, tome el 1 y multiplíquelo por 10, y luego sume el 5".

Si el número (o, mejor dicho, el **numeral**) tiene más dígitos, esos dígitos van multiplicados por **potencias de 10** que van creciendo hacia la izquierda. La cifra de las unidades está multiplicada por la potencia de 10 de exponente 0 (es decir, por 10^0, que es igual a 1).

Esto se cumple para todos los sistemas posicionales, sólo que con potencias **de la base correspondiente**.

Sistema binario

Comprender y manejar la notación en sistema binario es sumamente importante para el estudio de la computación. El sistema binario comprende únicamente dos dígitos, **0 y 1**.

Igual que ocurre con el sistema decimal, los numerales se escriben como suma de dígitos del sistema multiplicados por potencias de la base. En este sistema, cada 1 en una posición indica que sumamos una potencia de 2. Esa potencia de 2 es 2^n, donde n es la posición, y las posiciones se cuentan desde 0.

Por ejemplo,

$$10 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 1010_{(2}$$

y

$$15 = 1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 1111_{(2}$$

Trucos para conversión rápida

Las computadoras digitales, tal como las conocemos hoy, almacenan todos sus datos en forma de números binarios. Es **muy recomendable**, para la práctica de esta materia, adquirir velocidad y seguridad en la conversión de y a sistema binario.

Una manera de facilitar esto es memorizar los valores de algunas potencias iniciales de la base 2:

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \\ 2^5 &= 32 \\ 2^6 &= 64 \\ 2^7 &= 128 \end{aligned}$$

¿Qué utilidad tiene memorizar esta tabla? Que nos permite convertir mentalmente algunos casos simples de números en sistema decimal, a base 2. Por ejemplo, el número **40** equivale a **32 + 8**, que son ambas potencias de 2. Luego, la expresión de 40 en sistema binario será **101000**.

Otro truco interesante consiste en ver que si un numeral está en base 2, **multiplicarlo por 2 equivale a desplazar un lugar a la izquierda todos sus dígitos, completando con un 0 al final**. Así, si sabemos que $40_{(10} = 101000_{(2}$, ¿cómo escribimos rápidamente **80**, que es 40×2? Tomamos la expresión de 40 en base 2 y la desplazamos a la izquierda agregando un 0: $1010000_{(2} = 80_{(10}$.

De todas maneras, estos no son más que trucos que pueden servir en no todos los casos. Más adelante veremos el procedimiento de conversión general correcto.

Preguntas

- ¿Cuál es el truco para calcular rápidamente la expresión binaria de 20, si conocemos la de 40?
- ¿Cómo calculamos la de 40, si conocemos la de 10?
- ¿Cómo podemos expresar estas reglas en forma general?

Sistema hexadecimal

Otro sistema de numeración importante es el hexadecimal o de base 16. En este sistema tenemos **más dígitos** que en el decimal, por lo cual tenemos que recurrir a "dígitos" nuevos, tomados del alfabeto. Así, A representa el 10, B el 11, etc.

El sistema hexadecimal nos resultará útil porque con él podremos expresar fácilmente números que llevarían muchos dígitos en sistema binario.

- La conversión entre binarios y hexadecimales es sumamente directa.
- Al ser un sistema con más dígitos que el binario, la expresión de cualquier número será más corta.

Una expresión general

Como hemos visto intuitivamente en el sistema de contar con los dedos, y como hemos confirmado repasando los sistemas decimal, binario y hexadecimal, los sistemas posicionales tienen una cosa muy importante en común: las cifras de un **numeral** escrito en cualquier base no son otra cosa que los **factores por los cuales hay que multiplicar las sucesivas potencias de la base** para saber a qué **número** nos estamos refiriendo.

Por ejemplo, el numeral **2017** escrito en base 10 no es otra cosa que la suma de:

$$2 \times 1000 + 0 \times 100 + 1 \times 10 + 7 \times 1 =$$
$$2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$

Los dígitos 2, 0, 1 y 7 multiplican, respectivamente, a 10^3, 10^2, 10^1 y 10^0, que son potencias de la base 10. Este **numeral** designa al **número** 2017 porque esta cuenta, efectivamente, da **2017**.

Sin embargo, si el número está expresado en otra base, la cuenta debe hacerse con potencias de esa otra base. Si hablamos de $2017_{(8}$, entonces las cifras 2, 0, 1 y 7 multiplican a 8^3, 8^2, 8^1 y 8^0. Este **numeral** designa al **número** 1039 porque esta cuenta, efectivamente, da **1039**.

Este análisis permite enunciar una ley o expresión general que indica cómo se escribe un número n cualquiera, no negativo, en una base b:

$$n = x_k \times b^k + \ldots + x_2 \times b^2 + x_1 \times b^1 + x_0 \times b^0$$

Esta ecuación puede escribirse más sintéticamente en notación de sumatoria como:

$$n = \sum_{i=0}^{k} x_i \times b^i$$

En estas ecuaciones (que son equivalentes):

- Los números x_i son las cifras del numeral.
- Los números b^i son potencias de la base, cuyos exponentes crecen de derecha a izquierda y comienzan por 0.
- Las potencias están **ordenadas y completas**, y son tantas como las cifras del numeral.
- Los números x_i son necesariamente **menores que** b, ya que son dígitos en un sistema de numeración que tiene b dígitos.

Conversión de base

Veremos algunos casos interesantes de conversiones de base. Serán especialmente importantes los casos donde el número de origen o de destino de la conversión esté en base 10, nuestro sistema habitual, pero también nos dedicaremos a algunas conversiones de base donde ninguna de ellas sea 10.

Conversión de base 10 a otras bases

El procedimiento para convertir un número escrito en base 10 a cualquier otra base (llamémosla **base destino**) es siempre el mismo y se basa en la división entera (sin decimales):

- Dividir el número original por la base destino, anotando cociente y resto
- Mientras se pueda seguir dividiendo:
  - Volver al paso anterior reemplazando el número original por el nuevo cociente
- Finalmente escribimos los dígitos de nuestro número convertido usando **el último cociente y todos los restos en orden inverso a como aparecieron**.

Ésta es la expresión de nuestro número original en la base destino.

- Notemos que cada uno de los restos obtenidos es con toda seguridad **menor que la base destino**, ya que, en otro caso, podríamos haber seguido adelante con la división entera.

- Notemos también que el último cociente es también **menor que la base destino**, por el mismo motivo de antes (podríamos haber proseguido la división).
- Lo que acabamos de decir garantiza que tanto el último cociente, como todos los restos aparecidos en el proceso, **pueden ser dígitos de un sistema en la base destino** al ser todos menores que ella.

Pregunta

¿Cómo podemos usar la Expresión General para explicar por qué este procedimiento es correcto, al menos para el caso de convertir **61 a base 3**?

Conversión de otras bases a base 10

La conversión en el sentido opuesto, de una base b cualquiera a base 10, se realiza simplemente aplicando la Expresión General. Cada uno de los dígitos del número original (ahora en base b arbitraria) es el coeficiente de alguna potencia de la base original. Esta potencia depende de la posición de dicho dígito. Una vez que escribimos todos los productos de los dígitos originales por las potencias de la base, hacemos la suma y nos queda el resultado: el número original convertido a base 10.

Es de la mayor importancia cuidar de que las potencias de la base que intervienen en el cálculo estén **ordenadas y completas**. Es fácil si escribimos estas potencias a partir de la derecha, comenzando por la que tiene exponente 0, y vamos completando los términos de derecha a izquierda hasta agotar las posiciones del número original.

Preguntas

¿Cómo sería un sistema de **contar con los dedos en base 2**? Dedo arriba = 1, dedo abajo = 0...

- ¿Cómo hacemos el 1, el 2, el 3...?
- ¿Hasta qué número podemos contar con una mano? ¿Y con dos manos?
- ¿Y cómo se indica el **4** en este sistema?

Conversión entre bases arbitrarias

Hemos visto los casos de conversión entre base 10 y otras bases, en ambos sentidos. Ahora veamos los casos donde ninguna de las bases origen o destino son la base 10.

La buena noticia es que, en general, **esto ya sabemos hacerlo**. Si tenemos dos bases b_1 y b_2 cualesquiera, ninguna de las cuales es 10, sabiendo hacer las conversiones anteriores podemos hacer la conversión de b_1 a b_2 sencillamente haciendo **dos conversiones pasando por la base 10**. Si queremos convertir de b_1 a b_2, convertimos primero **de b_1 a base 10**, aplicando el procedimiento ya visto, y luego **de base 10 a b_2**. Eso es todo.

Pero en algunos casos especiales podemos aprovechar cierta relación existente entre las bases a convertir: por ejemplo, cuando son **2 y 16**, o **2 y 8**. La base 2 es la del sistema **binario**, y las bases 16 y 8 son las del sistema **hexadecimal** y del sistema **octal** respectivamente.

En estos casos, como 16 y 8 son potencias de 2 (la otra base), podemos aplicar un truco matemático para hacer la conversión en un solo paso y con muchísima facilidad. Por fortuna son estos casos especiales los que se presentan con mayor frecuencia en nuestra disciplina.

Equivalencias entre sistemas

Para poder aplicar este truco se necesita la tabla de equivalencias entre los dígitos de los diferentes sistemas. Si no logramos memorizarla, conviene al menos saber reproducirla, asegurándose de saber **contar** en las bases 2, 8 y 16 para reconstruir la tabla si es necesario. Pero con la práctica, se logra memorizarla fácilmente.

Notemos que:

- El sistema octal tiene ocho dígitos **(0 ... 7)** y cada uno de ellos se puede representar con **tres dígitos binarios**:
    - 000
    - 001
    - 010
    - 011
    - 100
    - 101
    - 110
    - 111

Notemos que:

- El sistema hexadecimal tiene dieciséis dígitos **(0 ... F)** y cada uno de ellos se puede representar con **cuatro dígitos binarios**:
    - 0000
    - 0001
    - 0010
    - 0011
    - 0100
    - 0101
    - 0110

- 0111
- 1000
- 1001
- 1010
- 1011
- 1100
- 1101
- 1110
- 1111

Conversión entre sistemas binario y hexadecimal

El truco para convertir de base 2 a base 16 consiste simplemente en agrupar los dígitos binarios de a cuatro, y reemplazar cada grupo de cuatro dígitos por su equivalente en base 16 según la tabla anterior.

Si hace falta completar un grupo de cuatro dígitos binarios, se completa con ceros a la izquierda.

Si el problema es convertir, inversamente, de base 16 a base 2, de igual forma reemplazamos cada dígito hexadecimal por los cuatro dígitos binarios que lo representan.

Conversión entre sistemas binario y octal

El problema de convertir entre bases 2 y 8 es igual de sencillo. Basta con reemplazar cada grupo de **tres** dígitos binarios (completando con ceros a la izquierda si hace falta) por el dígito octal equivalente. Lo mismo si la conversión es en el otro sentido.

Unidades de Información

En este segundo tema de la unidad veremos qué es la información y cómo podemos cuantificar, es decir, medir, la cantidad de información que puede alojar un dispositivo, o la cantidad de información que representa una pieza cualquiera de información. Veremos además las relaciones entre las diferentes unidades de información.

Información

A lo largo de la historia se han inventado y fabricado máquinas, que son dispositivos que **transforman la energía**, es decir, convierten una forma de energía en otra. Las computadoras, en cambio, convierten una forma de **información** en otra.

Los programas de computadora reciben alguna forma de información (la **entrada** del programa), la **procesan** de alguna manera, y emiten alguna información de **salida**.

La **entrada** es un conjunto de datos de partida, para que trabaje el programa, y la **salida** generada por el programa es alguna forma de respuesta o solución a un problema. Sabemos, además, que el material con el cual trabajan las computadoras son textos, documentos, mensajes, imágenes, sonido, etc. Todas estas son formas en las que se codifica y se almacena la información.

Un epistemólogo dice que la información es "una diferencia relevante". Si vemos que el semáforo cambia de rojo a verde, recibimos información ("podemos avanzar"). Al cambiar el estado del semáforo aparece una **diferencia** que puedo observar. Es **relevante** porque modifica de alguna forma el estado de mi conocimiento o me permite tomar una decisión respecto de algo.

¿Qué es, exactamente, esta información? No podemos tocarla ni pesarla, pero ¿se puede medir? Y si se puede medir, ¿entonces se puede medir la cantidad de información que aporta un texto, una imagen?

Bit

La Teoría de la Información, una teoría matemática desarrollada alrededor de 1950, dice que el **bit** es "la mínima unidad de información". Un bit es la información que recibimos "cuando se especifica una de dos alternativas igualmente probables". Si tenemos una pregunta **binaria**, es decir, aquella que puede ser respondida **con un sí o con un no**, entonces, al recibir una respuesta, estamos recibiendo un bit de información. Las preguntas binarias son las más simples posibles (porque no podemos decidir entre **menos** respuestas), de ahí que la información necesaria para responderlas sea la mínima unidad de información.

De manera que un bit es una unidad de información que puede tomar sólo dos valores. Podemos pensar estos valores como **verdadero o falso**, como **sí o no**, o como **0 y 1**.

Cuando las computadoras trabajan con piezas de información complejas, como los textos o imágenes, estas piezas son representadas como conjuntos ordenados de bits, de un cierto tamaño. Así, por ejemplo, la secuencia de ocho bits **01000001** puede representar la letra A mayúscula. Un documento estará constituido por palabras; éstas están formadas por símbolos como las letras, y éstas serán representadas por secuencias de bits.

La memoria de las computadoras está diseñada de forma que **no se puede almacenar otra cosa que bits** en esa memoria. Los textos, las imágenes, los sonidos, los videos, los programas que ejecuta, los mensajes que recibe o envía; todo lo que puede guardar, procesar, o emitir una computadora digital, debe estar en algún momento representado por una secuencia de bits. Los bits son, en cierta forma, como los átomos de la información. Por eso el bit es la unidad fundamental que usamos para medirla, y definiremos también algunas unidades mayores, o múltiplos.

El viaje de un bit

En una famosa película de aventuras hay una ciudad en problemas. Uno de los héroes enciende una pila de leña porque se prepara un terrible ataque sobre la ciudad. La pila de leña es el dispositivo preestablecido que tiene la ciudad para pedir ayuda en caso de emergencia.

En la cima de la montaña que está cruzando el valle existe un puesto similar, con su propio montón de leña, y un vigía. El vigía ve el fuego encendido en la ciudad que pide ayuda, y a su vez enciende su señal. Lo mismo se repite de cumbre en cumbre, atravesando grandes distancias en muy poco tiempo, hasta llegar rápidamente a quienes están en condiciones de prestar la ayuda. En una tragedia griega se dice que este ingenioso dispositivo se utilizó en la realidad, para comunicar en tan sólo una noche la noticia de la caída de Troya.

La información que está transportando la señal que viaja es la respuesta a una pregunta muy sencilla: **"¿la ciudad necesita nuestra ayuda?"**.

Esta pregunta es **binaria**: se responde con un sí o con un no. Por lo tanto, lo que ha viajado es **un bit de información**.

Notemos que, en los manuales de lógica o de informática, encontraremos siempre asociados los **bits** con los valores de **0 y 1**. Aunque esto es útil a los efectos de emplear los bits en computación, no es del todo exacto. Un bit no es exactamente un dígito. Lo que viajó desde la ciudad sitiada hasta su destino no es un 0 ni un 1. Es **un bit de información**, aquella unidad de información que permite tomar una decisión entre dos alternativas. Sin embargo, la identificación de los bits con los dígitos binarios es útil para todo lo que tiene que ver con las computadoras.

Byte

Como el bit es una medida tan pequeña de información, resulta necesario definir unidades más grandes. En particular, y debido a la forma como se organiza la memoria de las computadoras, es útil tener como unidad al **byte** (abreviado **B** mayúscula), que es una secuencia de **8 bits**. Podemos imaginarnos la memoria de las computadoras como una estantería muy alta, compuesta por estantes que contienen ocho casilleros. Cada uno de estos estantes es una **posición o celda de memoria**, y contiene exactamente ocho bits (un byte) de información.

Como los valores de los bits que forman un byte son independientes entre sí, existen 2^8 diferentes valores para esos ocho bits. Si los asociamos con números en el sistema binario, esos valores serán **00000000**, **00000001**, **00000010**, ..., etc., hasta el **11111111**. En decimal, esos valores corresponden a los números **0, 1, 2, ..., 255**.

Cada byte de la memoria de una computadora, entonces, puede alojar un número entre 0 y 255. Esos números representarán diferentes piezas de información: si los vemos como bytes independientes, pueden representar **caracteres** como letras y otros símbolos, pero también pueden estar formando parte de otras estructuras de información más complejas, y tener otros significados.

Representando datos con bytes

Para poder tratar y comunicar la información, que está organizada en bytes, es necesario que exista una asignación fija de valores binarios a caracteres. Es decir, se necesita una **tabla de caracteres** que asigne un símbolo a cada valor posible entre 0 y 255.

La memoria de la computadora es como un espacio donde se almacenan temporariamente contenidos del tamaño de un byte. Si pudiéramos ver el contenido de una sección de la memoria mientras la computadora está trabajando, veríamos que cada byte tiene determinados contenidos binarios. Esos contenidos pueden codificar los caracteres de un mensaje, carácter por carácter.

Sabiendo que la memoria está organizada en bytes, es interesante saber qué capacidad tendrá la memoria de una computadora dada y qué tamaño tendrán las piezas de información que caben en esta memoria. Como la memoria de una computadora, y la cantidad de información que compone un mensaje, un programa, una imagen, etc., suelen ser muy grandes, utilizamos **múltiplos** de la unidad de memoria, que es el byte.

Existen en realidad dos sistemas diferentes de múltiplos: el **Sistema Internacional** y el **Sistema de Prefijos Binarios**. Las unidades de ambos sistemas son parecidas, pero no exactamente iguales.

Los dos sistemas difieren esencialmente en el factor de la unidad en los sucesivos múltiplos. En el caso del Sistema Internacional, todos los factores son alguna potencia de 1000. En el caso del Sistema de Prefijos Binarios, todos los factores son potencias de 1024.

Sistema Internacional

En el llamado Sistema Internacional, la unidad básica, el byte, se multiplica por potencias de 1000. Así, tenemos:

- El **kilobyte** (1000 bytes)
- El **megabyte** (1000×1000 bytes = 1000 kilobytes = un millón de bytes)
- El **gigabyte** ($1000 \times 1000 \times 1000$ bytes = mil megabytes = mil millones de bytes)
- El **terabyte** ($1000 \times 1000 \times 1000 \times 1000$ bytes = mil gigabytes = un billón de bytes), y otros múltiplos mayores como **petabyte, exabyte, zettabyte, yottabyte**.

Como puede verse, cada unidad se forma multiplicando la anterior por 1000.

Los símbolos de cada múltiplo, utilizados al especificar las unidades, son **k minúscula** para **kilo**, **M mayúscula** para **mega**, **G mayúscula** para **giga**, **T mayúscula** para **tera**, etc.

Sistema de Prefijos Binarios

En el llamado Sistema de Prefijos Binarios, el byte se multiplica por potencias de 2^{10}, que es 1024. Así, tenemos:

- El **kibibyte** (1024 bytes)
- El **mebibyte** (1024 $\times$ 1024 bytes, **aproximadamente** un millón de bytes, pero exactamente 1048576 bytes)
- El **gibibyte** (1024 $\times$ 1024 $\times$ 1024 bytes, **aproximadamente** mil millones de bytes)
- El **tebibyte** (1024 $\times$ 1024 $\times$ 1024 $\times$ 1024 bytes, aproximadamente un millón de mebibytes, o aproximadamente un billón de bytes), y otros múltiplos mayores como **pebibyte, exbibyte, zebibyte, yobibyte**.

Como puede verse, cada unidad se forma multiplicando la anterior por 1024.

Notemos que los prefijos **kilo, mega, giga, tera**, del Sistema Internacional, cambian a **kibi, mebi, gibi, tebi**, del sistema de Prefijos Binarios.

Los símbolos de cada múltiplo, utilizados al especificar las unidades, son **Ki**, con K mayúscula, para **kibi**, **Mi** para **mebi**, **Gi** para **gibi**, **Ti** para **tebi**, etc.

¿Por qué existen **dos** sistemas en lugar de uno? En realidad la adopción del Sistema de Prefijos Binarios se debe a las características de la memoria de las computadoras:

- Cada posición o celda de la memoria tiene su dirección, que es el número de la posición de esa celda dentro del conjunto de toda la memoria de la computadora.
- Cuando la computadora accede a una posición o celda de su memoria, para leer o escribir un contenido en esa posición, debe especificar la dirección de la celda.
- Como la computadora usa exclusivamente números binarios, al especificar la dirección de la celda usa una cantidad de dígitos binarios.
- Por lo tanto, la cantidad de posiciones que puede acceder usando direcciones es una potencia de 2: si usa 8 bits para especificar cada dirección, accederá a 2^8 bytes, cuyas direcciones estarán entre 0 y 255. Si usa 10 bits, accederá a 2^{10} bytes, cuyas direcciones serán 0 a 1023.
- Entonces, tener una memoria de, por ejemplo, exactamente **mil bytes**, complicaría técnicamente las cosas porque las direcciones 1000 a 1023 no existirían. Si un programa quisiera acceder a la posición 1020 habría un grave problema.

Habría que tener en cuenta excepciones por todos lados y la vida de los diseñadores de computadoras y de los programadores sería lamentable.

- En consecuencia, todas las memorias se fabrican en tamaños que son potencias de 2 y el Sistema de Prefijos Binarios se adapta perfectamente a medir esos tamaños.

En computación se utilizan, en diferentes situaciones, ambos sistemas de unidades. Es costumbre usar el Sistema Internacional para hablar de velocidades de transmisión de datos o tamaños de archivos, pero usar Prefijos Binarios al hablar de almacenamiento de memoria, o en unidades de almacenamiento permanente, como los discos.

- Entonces, cuando un proveedor de servicios de Internet ofrece **un enlace de 1 Mbps**, nos está diciendo que por ese enlace podremos transferir **exactamente 1 millón de bits por segundo**. El proveedor utiliza el Sistema Internacional.
- Los textos, imágenes, sonido, video, programas, etc., se guardan en **archivos**, que son sucesiones de bytes. Encontramos archivos en el disco de nuestra computadora, y podemos descargar archivos desde las redes. Cuando nos interesa saber cuánto mide un archivo, en términos de bytes, usamos el Sistema Internacional porque el archivo no tiene por qué tener un tamaño que sea potencia de 2.
- Por el contrario, los fabricantes de medios de almacenamiento, como memorias, discos rígidos o pendrives, deberían (aunque a veces no lo hacen) utilizar Prefijos Binarios para expresar las capacidades de almacenamiento de esos medios. Así, un *"pendrive de dieciséis gigabytes"*, si tiene una capacidad de 16×2^{30} bytes, debería publicitarse en realidad como *"pendrive de dieciséis gibibytes"*.

Representación de datos numéricos

Veremos de qué manera puede ser tratada mediante computadoras la información correspondiente a números, textos, imágenes y otros datos. Necesitaremos conocer las formas de representación de datos, y comenzaremos por los datos numéricos.

Representación de datos numéricos

Hemos visto ejemplos de sistemas de numeración: en base 6, en base 10, o decimal, en base 2, o binario, en base 16, o hexadecimal, y en base 8, u octal; y sabemos convertir la representación de un número en cada una de estas bases, a los sistemas en las demás bases. Sin embargo, aún nos falta considerar la representación numérica de varios casos importantes:

- Hemos utilizado estos sistemas para representar únicamente números **enteros**. Nos falta ver de qué manera representar números racionales, es decir aquellos que tienen una parte fraccionaria (los "decimales").

- Además estos enteros han sido siempre **no negativos**, es decir, sabemos representar únicamente el 0 y los naturales. Nos falta considerar los negativos.
- Por otra parte, no nos hemos planteado el problema de la **cantidad de dígitos**. Idealmente, un sistema de numeración puede usar infinitos dígitos para representar números arbitrariamente grandes. Si bien esto es matemáticamente correcto, las computadoras son objetos físicos que tienen unas ciertas limitaciones, y con ellas no es posible representar números de infinita cantidad de dígitos.

En esta parte de la unidad mostraremos sistemas de representación utilizados en computación que permiten tratar estos problemas.

Clasificación de los números

Es conveniente repasar la clasificación de los diferentes conjuntos de números y conocer las diferencias importantes entre éstos. Los títulos en el cuadro (tomado de Wikipedia) son referencias a los artículos enciclopédicos sobre cada uno de esos conjuntos.

- Números complejos
- Complejos
- Reales
- Racionales
- Enteros
- Naturales
- Uno
- Naturales primos
- Naturales compuestos
- Cero
- Enteros negativos
- Fraccionarios
- Fracción propia
- Fracción impropia
- Irracionales
- Irracionales algebraicos
- Trascendentes
- Imaginarios

Preguntas

- El **cero**, ¿es un natural?
- ¿Existen números naturales negativos? ¿Y racionales negativos?

- ¿Es correcto decir que un racional tiene una parte decimal que es, o bien finita, o bien periódica?
- ¿Puede haber dos expresiones diferentes para el mismo número, en el mismo sistema de numeración decimal?
- El número 0.9999... con 9 periódico, y el número 1, ¿son dos números diferentes o el mismo número? Si son diferentes, ¿qué número se encuentra entre ellos dos?
- El número 1 es a la vez natural y entero. ¿Por qué no puede haber un número que sea a la vez racional e irracional?
- ¿Por qué jamás podremos computar la sucesión completa de decimales de π?

Datos enteros

Veremos un sistema de representación de datos no negativos, llamado **sin signo**, y tres sistemas de representación de datos numéricos enteros, llamados **signo-magnitud**, **complemento a 2** y **notación en exceso**.

Datos fraccionarios

Para representar fraccionarios consideraremos los sistemas de **punto fijo** y **punto flotante**.

Rango de representación

Cada sistema de representación de datos numéricos tiene su propio **rango de representación** (que podemos abreviar **RR**), o intervalo de números representables. Ningún número fuera de este rango puede ser representado en dicho sistema. Conocer este intervalo es importante para saber con qué limitaciones puede enfrentarse un programa que utilice alguno de esos sistemas.

El rango de los números representados bajo un sistema está dado por sus **límites inferior y superior**, que definen qué zona de la recta numérica puede ser representada. Como ocurre con todo intervalo numérico cerrado, el rango de representación puede ser escrito como $[a, b]$, donde a y b son sus límites inferior y superior, respectivamente.

Por la forma en que están diseñados, algunos sistemas de representación sólo pueden representar números muy pequeños, o sólo positivos, o tanto negativos como positivos. En general, el RR **será más grande cuantos más dígitos binarios**, o bits, tenga el sistema. Sin embargo, el RR depende también de la forma como el sistema **utilice** esos dígitos binarios, ya que un sistema puede ser más o menos **eficiente** que otro en el uso de esos dígitos, aunque la cantidad de dígitos sea la misma en ambos sistemas.

Por lo tanto, decimos que el rango de representación depende a la vez de la **cantidad de dígitos** y de la **forma de funcionamiento** del sistema de representación.

Representación sin signo SS(k)

Consideremos primero qué ocurre cuando queremos representar números enteros **no negativos** (es decir, **positivos o cero**) sobre una cantidad fija de bits.

En el sistema **sin signo**, simplemente usamos el sistema binario de numeración, tal como lo conocemos, **pero limitándonos a una cantidad fija** de dígitos binarios o bits. Podemos entonces abreviar el nombre de este sistema como **SS(k)**, donde k es la cantidad fija de bits, o ancho, de cada número representado.

Rango de representación de SS(k)

¿Cuál será el rango de representación? El **cero** puede representarse, así que el límite inferior del rango de representación será 0. Pero ¿cuál será el límite superior? Es decir, si la cantidad de dígitos binarios en este sistema es k, ¿cuál es el número más grande que podremos representar?

Podemos estudiarlo de dos maneras.

1. Usando combinatoria

Contemos cuántos números diferentes podemos escribir con k dígitos binarios. Imaginemos un número binario cualquiera con k dígitos. El dígito de más a la derecha tiene únicamente dos posibilidades (0 o 1). Por cada una de éstas hay nuevamente dos posibilidades para el siguiente hacia la izquierda (lo que da las cuatro posibilidades 00, 01, 10, 11). Por cada una de éstas, hay dos posibilidades para el siguiente (dando las ocho posibilidades 000, 001, 010, 011, 100, 101, 110, 111), etc., y así hasta la posición k. No hay más posibilidades. Como hemos multiplicado 2 por sí mismo k veces, la cantidad de números que se pueden escribir es 2^k. Luego, el número más grande posible es $2^k - 1$. (**Pregunta**: ¿Por qué $2^k - 1$ y no 2^k?).

2. Usando álgebra

El número más grande que podemos representar en un sistema sin signo a k dígitos es, seguramente, aquel donde todos los k dígitos valen **1**. La Expresión General que hemos visto nos dice que si un número n está escrito en base 2, **con k dígitos**, entonces

$$n = x_{k-1} \times 2^{k-1} + ... + x_1 \times 2^1 + x_0 \times 2^0$$

y, si queremos escribir el más grande de todos, deberán ser todos los x_i iguales a 1. (**Pregunta**: ¿Por qué si el número n tiene k dígitos binarios, el índice del más significativo es $k-1$ y no k?)

Esta suma vale entonces

$$x_{k-1} \times 2^{k-1} + ... + x_1 \times 2^1 + x_0 \times 2^0 =$$
$$= 1 \times 2^{k-1} + ... + 1 \times 2^1 + 1 \times 2^0 =$$
$$= 2^{k-1} + ... + 2^1 + 2^0 =$$

$$= 2^k - 1$$

Usando ambos argumentos hemos llegado a que el número más grande que podemos representar con k dígitos binarios es $2^k - 1$. Por lo tanto, **el rango de representación de un sistema sin signo a k dígitos, o SS(k), es** $[0, 2^k - 1]$. Todos los números representables en esta clase de sistemas son **positivos o cero**.

Ejemplos

- Para un sistema de representación sin signo a 8 bits: $[0, 2^8 - 1] = [0, 255]$
- Con 16 bits: $[0, 2^{16} - 1] = [0, 65{,}535]$
- Con 32 bits: $[0, 2^{32} - 1] = [0, 4{,}294{,}967{,}295]$

Representación con signo

En la vida diaria manejamos continuamente números negativos, y los distinguimos de los positivos simplemente agregando un signo "menos". Representar esos datos en la memoria de la computadora no es tan directo, porque, como hemos visto, la memoria **solamente puede alojar ceros y unos**. Es decir, ¡no podemos simplemente guardar un signo "menos"! Lo único que podemos hacer es almacenar secuencias de ceros y unos.

Esto no era un problema cuando los números eran no negativos. Para poder representar, ahora, tanto números **positivos como negativos**, necesitamos cambiar la forma de representación. Esto quiere decir que una secuencia particular de dígitos binarios, que en un sistema sin signo tiene un cierto significado, ahora tendrá un significado diferente. Algunas secuencias, que antes representaban números positivos, ahora representarán negativos.

Veremos los **sistemas de representación con signo** llamados **Signo-magnitud (SM)**, **Complemento a 2 (C2)** y **Notación en exceso**.

Es importante tener en cuenta que **solamente se puede operar entre datos representados con el mismo sistema de representación**, y que el **resultado** de toda operación **vuelve a estar representado en el mismo sistema**.

Preguntas

- ¿Cuáles son los límites del rango de representación de un sistema de representación numérica?
- Un número escrito en un sistema de representación **con signo**, ¿es siempre negativo?
- ¿Para qué querríamos escribir un número positivo en un sistema de representación con signo?

Sistema de Signo-magnitud SM(k)

El sistema de **Signo-magnitud** no es el más utilizado en la práctica, pero es el más sencillo de comprender. Se trata simplemente de utilizar un bit (el de más a la izquierda) para representar el **signo**. Si este bit tiene valor 0, el número representado es positivo; si es 1, es negativo. Los demás bits se utilizan para representar la **magnitud**, es decir, el **valor absoluto** del número en cuestión.

Ejemplos

- $7_{(10} = 00000111_{(2}$
- $-7_{(10} = 10000111_{(2}$

Como estamos reservando un bit para expresar el signo, ese bit ya no se puede usar para representar magnitud; y como el sistema tiene una cantidad de bits fija, el RR ya no podrá representar el número máximo que era posible con el sistema **sin signo**.

Rango de representación de SM(k)

- En todo número escrito en el sistema de signo-magnitud a k bits, ya sea positivo o negativo, hay un bit reservado para el signo, lo que implica que quedan $k-1$ bits para representar su valor absoluto.
- Siendo un valor absoluto, estos $k-1$ bits representan un número **no negativo**. Además este número está representado con el sistema **sin signo** sobre $k-1$ bits, es decir, SS(k-1).
- Este número no negativo en SS(k-1) tendrá un valor máximo representable que coincide con el **límite superior** del rango de representación **de SS(k-1)**.
- Sabemos que el rango de representación de SS(k) es $[0, 2^k - 1]$. Por lo tanto, el rango de SS(k-1), reemplazando, será $[0, 2^{k-1} - 1]$.
- Esto quiere decir que el número representable en SM(k) cuyo valor absoluto es máximo, es $2^{k-1} - 1$. Por lo tanto éste es el límite superior del rango de representación de SM(k).
- Pero en SM(k) también se puede representar su opuesto negativo, simplemente cambiando el bit más alto por 1. El opuesto del máximo positivo representable es a su vez el número más pequeño, negativo, representable: $-(2^{k-1} - 1)$.

Con lo cual hemos calculado tanto el límite inferior como el superior del rango de representación de SM(k), que, finalmente, es $[-(2^{k-1} - 1), 2^{k-1} - 1]$.

Limitaciones de Signo-Magnitud

Si bien **SM(k)** es simple, no es tan efectivo, por varias razones:

- Existen dos representaciones del 0 ("positiva" y "negativa"), lo cual desperdicia un representante.
- Esto acorta el rango de representación.
- La aritmética en SM no es fácil, ya que cada operación debe comenzar por averiguar si los operandos son positivos o negativos, operar con los valores absolutos y ajustar el resultado de acuerdo al signo reconocido anteriormente.
- El problema aritmético se agrava con la existencia de las dos representaciones del cero: cada vez que un programa quisiera comparar un valor resultado de un cómputo con 0, debería hacer **dos** comparaciones.

Por estos motivos, el sistema de SM dejó de usarse y se diseñó un sistema que eliminó estos problemas, el sistema de **complemento a 2**.

Sistema de Complemento a 2

Para comprender el sistema de complemento a 2 es necesario primero conocer la **operación** de complementar a 2.

Operación de Complemento a 2

La **operación** de complementar a 2 consiste aritméticamente en obtener el **opuesto** de un número (el que tiene el mismo valor absoluto pero signo opuesto).

Para obtener el complemento a 2 de un número escrito en base 2, **se invierte cada uno de los bits (reemplazando 0 por 1 y viceversa) y al resultado se le suma 1**.

Otra forma

Otro modo de calcular el complemento a 2 de un número en base 2 es **copiar los bits, desde la derecha, hasta el primer 1 inclusive; e invertir todos los demás a la izquierda**.

Propiedad fundamental

El resultado de esta operación, C2(a), es el opuesto del número original a, y por lo tanto tiene la propiedad de que a y C2(a) suman 0:

$$C2(a) + a = 0$$

Comprobación

Podemos comprobar si la complementación fue bien hecha aplicando la **propiedad fundamental** del complemento. Si, al sumar nuestro resultado con el número original, no obtenemos 0, corresponde revisar la operación.

Ejemplos

- Busquemos el complemento a 2 de 111010. Invirtiendo todos los bits, obtenemos 000101. Sumando 1, queda 000110.
- Busquemos el complemento a 2 de 0011. Invirtiendo todos los bits, obtenemos 1100. Sumando 1, queda 1101.
- Comprobemos que el resultado obtenido en el último caso, 1101, es efectivamente el opuesto de 0011: $0011 + 1101 = 0$.

Representación en Complemento a 2

Ahora que contamos con la **operación de complementar a 2**, podemos ver cómo se construye el **sistema de representación en Complemento a 2**.

Para representar un número a en complemento a 2 a k bits, comenzamos por considerar su signo:

- Si a es positivo o cero, lo representamos como en SM(k), es decir, lo escribimos en base 2 a k bits.
- Si a es negativo, tomamos su valor absoluto y lo complementamos a 2.

Ejemplos

- Representemos el número 17 en complemento a 2 con 8 bits. Como es positivo, lo escribimos en base 2, obteniendo 00010001, que es 17 en notación complemento a 2 con 8 bits.
- Representemos el número -17 en complemento a 2 con 8 bits. Como es negativo, escribimos su valor absoluto en base 2, que es 00010001, y lo complementamos a 2. El resultado final es 11101111 que es -17 en notación complemento a 2 con 8 bits.

Conversión de C2 a base 10

Para convertir un número n, escrito en el sistema de complemento a 2, a decimal, lo primero es determinar el signo. Si el bit más alto es 1, n es negativo. En otro caso, n es positivo. Utilizaremos esta información enseguida.

- Si n es positivo, se interpreta el número como en el sistema sin signo, es decir, se utiliza la Expresión General para hacer la conversión de base como normalmente.
- Si n es negativo, se lo complementa a 2, obteniendo el opuesto de n. Este número, que ahora es positivo, se convierte a base 10 como en el caso anterior; y finalmente se le agrega el signo "-" para reflejar el hecho de que es negativo.

Ejemplos

- Convertir a decimal $n = 00010001$. Es positivo, luego, aplicamos la Expresión General dando $17_{(10}$.
- Convertir a decimal $n = 11101111$. Es negativo; luego, lo complementamos a 2 obteniendo 00010001. Aplicamos la Expresión General obteniendo $17_{(10}$. Como n era negativo, agregamos el signo menos y obtenemos el resultado final $-17_{(10}$.

RR de C2 con k bits

La forma de utilizar los bits en el sistema de complemento a 2 permite recuperar un representante que estaba desperdiciado en SM.

El rango de representación del sistema complemento a 2 sobre k bits es $[-(2^{k-1}), 2^{k-1} - 1]$. El límite superior del RR de C2 es el mismo que el de SM, pero el **límite inferior** es menor; luego el RR de C2 es mayor que el de SM.

El sistema de complemento a 2 tiene otras ventajas sobre SM:

- El cero tiene una única representación, lo que facilita las comparaciones.
- Las cuentas se hacen bit a bit, en lugar de requerir comprobaciones de signo.
- El mecanismo de cálculo es eficiente y fácil de implementar en hardware.
- Solamente se requiere diseñar un algoritmo para **sumar**, no uno para sumar y otro para restar.

Comparando rangos de representación

Diferentes sistemas, entonces, tienen diferentes rangos de representación. Si construimos un cuadro donde podamos comparar los rangos de representación **sin signo, signo-magnitud y complemento a 2** para una misma cantidad de bits, veremos que todas las combinaciones de bits están utilizadas, sólo que de diferente forma.

El cuadro comparativo para cuatro bits mostrará que las combinaciones 0000...1111 representan los primeros 16 números no negativos para el sistema sin signo, mientras que esas mismas combinaciones tienen otro significado en los sistemas con signo. En éstos últimos, una misma combinación con el bit más significativo en 1 siempre es negativa, pero el orden en que aparecen esas combinaciones es diferente entre SM y C2.

Por otro lado, los números positivos quedan representados por combinaciones idénticas en los tres sistemas, hasta donde lo permite el rango de representación de cada uno.

Si descartamos el bit de signo y consideramos sólo las magnitudes, los números negativos en SM aparecen con sus magnitudes crecientes alejándose del 0, mientras que en C2 esas magnitudes comienzan en cero al representar el negativo más pequeño posible y crecen a medida que se acercan al cero.

Aritmética en C2

Una gran ventaja que aporta el sistema en Complemento a 2 es que los diseñadores de hardware no necesitan implementar algoritmos de resta además de los de la suma. Cuando se necesita efectuar una resta, **se complementa el sustraendo** y luego se lo **suma** al minuendo. Las computadoras no restan: siempre suman.

Por ejemplo, la operación $9-8$ se realiza como $9+(-8)$, donde (-8) es el complemento a 2 de 8.

Preguntas

- Un número en complemento a 2, ¿tiene siempre su bit más a la izquierda en 1?
- El complemento a 2 de un número, es decir, **C2(x)**, ¿es siempre un número negativo?
- ¿Quién es **C2(0)**?
- ¿Cuánto vale **C2(C2(x))**? Es decir, ¿qué pasa si complemento a 2 el complemento a 2 de x?
- ¿Cuánto vale **x + C2(x)**? Es decir, ¿qué pasa si sumo a x su propio complemento a 2?
- ¿Cómo puedo verificar si calculé correctamente un complemento a 2?

Overflow o desbordamiento en C2

En todo sistema de ancho fijo, la suma de **dos números positivos, o de dos números negativos** puede dar un resultado que sea imposible de representar debido a las limitaciones del rango de representación. Este problema se conoce como desbordamiento, u *overflow*. Cuando ocurre una situación de overflow, el resultado de la operación **no es válido** y debe ser descartado.

Si conocemos los valores en decimal de dos números que queremos sumar, usando nuestro conocimiento del rango de representación del sistema podemos saber si el resultado quedará dentro de ese rango, y así sabemos, de antemano, si ese resultado será válido. Pero las computadoras no tienen forma de conocer a priori esta condición, ya que todo lo que tienen es la representación en C2 de ambos números. Por eso necesitan alguna forma de detectar las situaciones de overflow, y el modo más fácil para ellas es comprobar los dos últimos bits de la fila de bits de acarreo o *carry*.

El último bit de la fila de carry, el que se posiciona en la última de las k columnas de la representación, se llama *carry-in*. El siguiente bit de carry, el que ya no puede acarrearse sobre ningún dígito válido porque se han rebasado los k dígitos de la representación, se llama el *carry-out*.

- Si, luego de efectuar una suma en C2, los valores de los bits de *carry-in* y *carry-out* son **iguales**, entonces la computadora detecta que el resultado no ha desbordado y que **la suma es válida**. La operación de suma se ha efectuado exitosamente.

- Si, luego de efectuar una suma en C2, los valores de los bits de *carry-in* y *carry-out* son **diferentes**, entonces la computadora detecta que el resultado ha desbordado y que **la suma no es válida**. La operación de suma no se ha llevado a cabo exitosamente, y el resultado debe ser descartado.

Suma sin overflow

Siguiendo atentamente la secuencia de bits de carry podemos detectar, igual que lo hace la computadora, si se producirá un desbordamiento. En el caso de la operación $23 + (-9)$, el resultado (que es 14) cae dentro del rango de representación, y esto se refleja en los bits de *carry-in* y de *carry-out*, cuyos valores son iguales.

Suma con overflow

En el caso de la operación $123 + 9$ en C2 a 8 bits, el resultado (que es 132) cae fuera del rango de representación. Esto se refleja en los bits de *carry-in* y de *carry-out*, que son diferentes. El resultado no es válido y debe ser descartado.

Preguntas

- ¿Qué condición sobre los bits de carry permite asegurar que **no habrá** overflow?
- ¿Para qué sistemas de representación numérica usamos la condición de detección de overflow?
- ¿Puede existir overflow al sumar dos números de diferente signo?
- ¿Qué condición sobre los bits **de signo** de los operandos permite asegurar que **no habrá** overflow?
- ¿Puede haber casos de overflow al sumar dos números negativos?
- ¿Puede haber casos de overflow al restar dos números?

Extensión de signo en C2

Para poder efectuar una suma de dos números, ambos operandos deben estar representados en el mismo sistema de representación.

- Una suma de dos operandos donde uno esté, por ejemplo, en SM y el otro en C2, no tiene sentido aritmético.
- Además, la cantidad de bits de representación debe ser la misma.

En una suma en C2, si uno de los operandos estuviera expresado en un sistema con menos bits que el otro, será necesario convertirlo al sistema del otro (**extenderlo**) y operar con ambos en ese sistema de mayor ancho.

Si el operando en el sistema de menor ancho es positivo, la extensión se realiza simplemente **completando con ceros a la izquierda** hasta obtener la cantidad de dígitos del otro sistema. Si el operando del menor ancho es negativo, la extensión de signo se hace **agregando unos**.

Ejemplos

- $A + B = 00101011_{(2} + 00101_{(2}$
  - A está en C_2^8 y B en C_2^5
  - Se completa B (positivo) como $00000101_{(2}$
- $A + B = 1010_{(2} + 0110100_{(2}$
  - A está en C_2^4 y B en C_2^7
  - Se completa A (negativo) como $1111010_{(2}$

Notación en exceso o *bias*

En un sistema de notación en exceso, se elige un intervalo $[a, b]$ de enteros a representar, y todos los valores dentro del intervalo se representan con una secuencia de bits de la misma longitud.

La cantidad de bits deberá ser la necesaria para representar todos los enteros del intervalo, inclusive los límites, y por lo tanto estará en función de la longitud del intervalo. Un intervalo $[a, b]$ de enteros, con sus límites incluidos, comprende exactamente $n = b - a + 1$ valores. Esta longitud del intervalo debe ser cubierta con una cantidad k de bits suficiente, lo cual obliga a que $2^k \geq n$. Supongamos que n sea una potencia de 2 para facilitar las ideas, de forma que $2^k = n$.

Las 2^k secuencias de k bits, ordenadas como de costumbre según su valor aritmético, se aplican a los enteros en $[a, b]$, uno por uno. Es decir, si usamos 3 bits, las secuencias serán 000, 001, 010, ... hasta 111; y los valores representados serán respectivamente:

- 000 $= a$
- 001 $= a + 1$
- 010 $= a + 2$
- ...
- 111 $= b$

Notemos que tanto a como b pueden ser **negativos**. Así podemos representar intervalos de enteros arbitrarios con secuencias de k bits, lo que nos vuelve a dar un sistema de representación con signo.

Con este método no es necesario que el bit de orden más alto represente el signo. Tampoco que el intervalo contenga la misma cantidad de números negativos que positivos o cero, aunque para la mayoría de las aplicaciones es lo más razonable.

El sistema en exceso se utiliza como componente de otro sistema de representación más complejo, la representación en punto flotante.

Conversión entre exceso y decimal

Una vez establecido un sistema en exceso que representa el intervalo $[a, b]$ en k bits:

- Para calcular la secuencia binaria que corresponde a un valor decimal d, a d **le restamos** a y luego convertimos el resultado (que será **no negativo**) a **SS(k)**, es decir, a binario sin signo sobre k bits.
- Para calcular el valor decimal d representado por una secuencia binaria, convertimos la secuencia a decimal como en **SS(k)**, y al resultado (que será **no negativo**) le **sumamos** el valor de a.

Ejemplos

Representemos en sistema en exceso el intervalo $[10, 25]$ (que contiene $25-10+1 = 16$ enteros). Como necesitamos 16 secuencias binarias, usaremos 4 bits que producirán las secuencias 0000, 0001, ..., 1111.

- Para calcular la secuencia que corresponde al número 20, hacemos $20-10 = 10$ y el resultado será la secuencia **1010**.
- Para calcular el valor decimal que está representando la secuencia **1011**, convertimos 1011 a decimal, que es 11, y le sumamos 10; el resultado es 21.

Representemos en sistema en exceso el intervalo $[-3, 4]$ (que contiene $4-(-3)+1 = 8$ enteros). Como necesitamos 8 secuencias binarias, usaremos 3 bits que producirán las secuencias 000, 001, ..., 111.

- Para calcular la secuencia que corresponde al número 2, hacemos $2-(-3) = 5$ y el resultado será la secuencia **101**.
- Para calcular el valor decimal que está representando la secuencia **011**, convertimos 011 a decimal, que es 3, y le sumamos -3; el resultado es 0.

Preguntas sobre Notación en Exceso

- Dado un valor decimal a representar, ¿cómo calculamos el binario?
- Dado un binario, ¿cómo calculamos el valor decimal representado?
- El sistema en exceso ¿destina un bit para representar el signo?
- ¿Se puede representar un intervalo que no contenga el cero?
- ¿Cómo se comparan dos números en exceso para saber cuál es el mayor?

Continuará