Ejercicio 1. Indique la representación de los siguientes números, razonando su respuesta:

a) -16 en complemento a 2 con 5 bits

b) -16 en complemento a 1 con 5 bits

c) +13 en signo magnitud con 5 bits

d) -14 en complemento a dos con 5 bits

Solución:

a) El rango de representación de números en complemento a dos con 5 bits es [-25-1..25-1-1] = [-16..15]. 16 en

binario es 10000. Tenemos que complementar, 01111 y sumarle 1. Por tanto, -16 en complemento a dos con

5 bits es 10000.

b) El rango de representación de números en complemento a dos con 5 bits es [-25-1..25-1-1] = [-15..15]. Por

tanto, el número -16 no se puede representar.

c) 13 en binario puro es 1101. En signo magnitud se introduce al comienzo un bit de signo, en este caso 0 para

indicar que es positivo. El número 13 es signo magnitud con 5 bits es 01101.

d) 14 en binario puro con 5 bits es 01110. Se complementa, 10001 y se suma 1, con lo que se obtiene 10010.la representación de números en diferentes sistemas numéricos: complemento a uno, complemento a dos y signo magnitud. Este código es especialmente útil para comprender cómo se representan los números negativos en sistemas binarios y los diferentes enfoques utilizados para ello.}

1. En la clase **RepresentacionNumeros**, se encuentra el método principal **main** que es el punto de entrada del programa. Aquí se definen los números a ser representados y se invocan las funciones correspondientes para obtener sus representaciones.
2. Para cada número a ser representado, se declara una variable entera (**a**, **b**, **c**, **d**) y se les asigna el valor correspondiente.
3. A continuación, se invocan las funciones de representación (**complementoUno**, **complementoDos**, **signoMagnitud**) pasando como argumentos el número y la cantidad de bits requeridos.
4. Dentro de cada función de representación, se realiza la verificación del rango de representación. Para ello, se calculan los valores máximo y mínimo permitidos según la cantidad de bits especificada. Si el número se encuentra dentro de ese rango, se procede a realizar la representación correspondiente.
5. a función **complementoUno** calcula el complemento a uno de un número. Se verifica si el número es negativo, y si es así, se toma el valor absoluto y se calcula su representación binaria. Luego, se completa con ceros a la izquierda hasta alcanzar la cantidad de bits requeridos. Si el número original era negativo, se aplica la operación de complemento a uno invirtiendo todos los bits.

“*public static String complementoUno(int number, int bits) {:*

*Esta línea define una función llamada complementoUno que recibe dos parámetros: number (el número a representar) y bits (el número de bits para la representación).*

*int max = (int) Math.pow(2, bits - 1) - 1;:*

*Calcula el valor máximo que puede ser representado en complemento a uno con el número de bits dado. Se utiliza la fórmula 2^(bits-1) - 1 para obtener este valor.*

*int min = -max;:*

*Calcula el valor mínimo que puede ser representado en complemento a uno. Es simplemente el negativo del valor máximo calculado anteriormente.*

*if (number >= min && number <= max) {:*

*Verifica si el número dado está dentro del rango representable en complemento a uno con los bits proporcionados. Si el número cumple con esta condición, se procede a calcular su representación.*

*StringBuilder binary = new StringBuilder(Integer.toBinaryString(Math.abs(number)));:*

*Convierte el valor absoluto del número dado en una representación binaria utilizando Integer.toBinaryString(). El resultado se almacena en un objeto StringBuilder llamado binary.*

*while (binary.length() < bits) {:*

*Verifica si la longitud de la representación binaria es menor que el número de bits especificados. Si es así, se agregan ceros a la izquierda para completar los bits faltantes.*

*binary.insert(0, "0");:*

*Inserta un cero al principio de la representación binaria para completar los bits faltantes a la izquierda. Esto se realiza dentro de un bucle while hasta que la longitud de la representación binaria sea igual al número de bits especificados.*

*if (number < 0) {:*

*Comprueba si el número original es negativo.*

*binary = complement(binary);:*

*Si el número es negativo, se llama a la función auxiliar complement() para obtener el complemento a uno de la representación binaria.*

*return binary.toString();:*

*Devuelve la representación binaria, ya sea la original o el complemento a uno, en forma de cadena.*

*} else {:*

*Si el número dado está fuera del rango representable, se ejecuta este bloque de código.*

*return "El número no es representable en " + bits + " bits";:*

*Devuelve un mensaje indicando que el número no puede ser representado con el número de bits proporcionados.”*

1. La función **complementoDos** calcula el complemento a dos de un número. Si el número es positivo, se calcula su representación binaria y se completa con ceros a la izquierda hasta alcanzar la cantidad de bits requeridos. Si el número es negativo, se calcula su complemento a dos sumándole el valor de 2 elevado a la cantidad de bits. Luego, se toma la representación binaria de ese complemento y se obtienen los últimos **bits** dígitos, descartando los bits adicionales.

“

* **public static String complementoDos(int number, int bits) {**:
  + Esta línea define una función llamada **complementoDos** que recibe dos parámetros: **number** (el número a representar) y **bits** (el número de bits para la representación).
* **int max = (int) Math.pow(2, bits - 1) - 1;**:
  + Calcula el valor máximo que puede ser representado en complemento a dos con el número de bits dado. Se utiliza la fórmula 2^(bits-1) - 1 para obtener este valor.
* **int min = -max - 1;**:
  + Calcula el valor mínimo que puede ser representado en complemento a dos. Es el negativo del valor máximo calculado anteriormente, restándole 1 adicionalmente.
* **if (number >= min && number <= max) {**:
  + Verifica si el número dado está dentro del rango representable en complemento a dos con los bits proporcionados. Si el número cumple con esta condición, se procede a calcular su representación.
* **if (number >= 0) {**:
  + Comprueba si el número original es mayor o igual a cero.
* **StringBuilder binary = new StringBuilder(Integer.toBinaryString(number));**:
  + Convierte el número dado en una representación binaria utilizando **Integer.toBinaryString()**. El resultado se almacena en un objeto **StringBuilder** llamado **binary**.
* **while (binary.length() < bits) {**:
  + Verifica si la longitud de la representación binaria es menor que el número de bits especificados. Si es así, se agregan ceros a la izquierda para completar los bits faltantes.
* **binary.insert(0, "0");**:
  + Inserta un cero al principio de la representación binaria para completar los bits faltantes a la izquierda. Esto se realiza dentro de un bucle while hasta que la longitud de la representación binaria sea igual al número de bits especificados.
* **return binary.toString();**:
  + Devuelve la representación binaria en forma de cadena.
* **} else {**:
  + Si el número original es negativo, se ejecuta este bloque de código.
* **int complement = (int) Math.pow(2, bits) + number;**:
  + Calcula el complemento a dos del número dado sumándole 2^bits. Esto se hace para obtener la representación en complemento a dos del número negativo.
* **String binary = Integer.toBinaryString(complement);**:
  + Convierte el complemento a dos obtenido en una representación binaria utilizando **Integer.toBinaryString()**. El resultado se almacena en una variable **binary** de tipo **String**.
* **return binary.substring(binary.length() - bits);**:
  + Devuelve una subcadena de la representación binaria que tiene una longitud igual al número de bits especificados. Esto se hace para asegurarse de que se cumpla el número de bits requeridos.
* **} else {**:
  + Si el número dado está fuera del rango representable, se ejecuta este bloque de código.
* **return "El número no es representable en " + bits + " bits";**:
  + Devuelve un mensaje indicando que el número no puede ser representado con el número de bits proporcionados.

”

1. La función **signoMagnitud** calcula la representación en signo magnitud de un número. Se verifica si el número es negativo, y si es así, se toma el valor absoluto y se calcula su representación binaria. Luego, se completa con ceros a la izquierda hasta alcanzar la cantidad de bits requeridos. Se agrega un bit adicional al comienzo para indicar el signo, siendo "0" para números positivos y "1" para números negativos.

“

* ***public static String signoMagnitud(int number, int bits) {****:*
  + *Esta línea define una función llamada* ***signoMagnitud*** *que recibe dos parámetros:* ***number*** *(el número a representar) y* ***bits*** *(el número de bits para la representación).*
* ***int max = (int) Math.pow(2, bits - 1) - 1;****:*
  + *Calcula el valor máximo que puede ser representado en signo magnitud con el número de bits dado. Utiliza la fórmula 2^(bits-1) - 1 para obtener este valor.*
* ***int min = -max;****:*
  + *Calcula el valor mínimo que puede ser representado en signo magnitud. Es el negativo del valor máximo calculado anteriormente.*
* ***if (number >= min && number <= max) {****:*
  + *Verifica si el número dado está dentro del rango representable en signo magnitud con los bits proporcionados. Si el número cumple con esta condición, se procede a calcular su representación.*
* ***StringBuilder binary = new StringBuilder(Integer.toBinaryString(Math.abs(number)));****:*
  + *Convierte el valor absoluto del número dado en una representación binaria utilizando* ***Integer.toBinaryString()****. El resultado se almacena en un objeto* ***StringBuilder*** *llamado* ***binary****.*
* ***while (binary.length() < bits - 1) {****:*
  + *Verifica si la longitud de la representación binaria es menor que el número de bits especificados menos uno. Si es así, se agregan ceros a la izquierda para completar los bits faltantes (excepto el bit de signo).*
* ***binary.insert(0, "0");****:*
  + *Inserta un cero al principio de la representación binaria para completar los bits faltantes a la izquierda (excepto el bit de signo). Esto se realiza dentro de un bucle while hasta que la longitud de la representación binaria sea igual al número de bits especificados menos uno.*
* ***if (number < 0) {****:*
  + *Comprueba si el número original es negativo.*
* ***binary.insert(0, "1");****:*
  + *Si el número es negativo, se inserta un uno al principio de la representación binaria para indicar el signo negativo.*
* ***else {****:*
  + *Si el número original es positivo, se ejecuta este bloque de código.*
* ***binary.insert(0, "0");****:*
  + *Si el número es positivo, se inserta un cero al principio de la representación binaria para indicar el signo positivo.*
* ***return binary.toString();****:*
  + *Devuelve la representación binaria en forma de cadena.*
* ***} else {****:*
  + *Si el número dado está fuera del rango representable, se ejecuta este bloque de código.*
* ***return "El número no es representable en " + bits + " bits";****:*
  + *Devuelve un mensaje indicando que el número no puede ser representado con el número de bits proporcionados.*

”

1. Finalmente, se imprime por pantalla el resultado de cada representación obtenida.

* “**public static StringBuilder complement(StringBuilder binary) {**:
  + Esta línea define una función llamada **complement** que toma un objeto **StringBuilder** llamado **binary** como argumento. La función calculará el complemento de este número binario.
* **for (int i = 0; i < binary.length(); i++) {**:
  + Inicia un bucle **for** que recorre cada posición del número binario, representada por el índice **i**.
* **if (binary.charAt(i) == '0') {**:
  + Comprueba si el dígito en la posición **i** del número binario es igual a '0'.
* **binary.setCharAt(i, '1');**:
  + Si el dígito es '0', se cambia a '1' utilizando el método **setCharAt()** de **StringBuilder**. Esto implica complementar ese dígito.
* **} else {**:
  + Si el dígito en la posición **i** no es '0' (es '1'), se ejecuta este bloque de código.
* **binary.setCharAt(i, '0');**:
  + Se cambia el dígito '1' en la posición **i** a '0'. Esto también implica complementar ese dígito.
* **}**:
  + Finaliza el bloque **else** y el bucle **for**.
* **return binary;**:
  + Devuelve el objeto **StringBuilder** **binary** que contiene el número binario complementado.

”