



DCC
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 1 - Representaciones Numéricas I

Profesor: Germán Leandro Contreras Sagredo

Objetivos de la clase

- Entender la utilidad del uso de números para la representación de datos en el computador.
- Conocer distintas representaciones de números naturales y enteros.
- Realizar ejercicios que consoliden los conocimientos anteriores.

Representación de datos

A través de los números, podemos representar distintos tipos de dato:

- Caracteres (ASCII - estándar que asocia números a caracteres).
- Imagen: Conjunto de pixeles con representación numérica (color).
- Sonido: Secuencia numérica (muestra) que se *mapea* a distintas amplitudes en un intervalo de tiempo.
- Video: Secuencia de imágenes con sonido.


Representación de datos




En resumen, sabiendo representar números dentro del computador, podemos representar una amplia gama de datos en este.

Ahora... **¿cómo lo hacemos?**










Representaciones de números naturales


Asumamos que volvemos a una época sin computadores, papel ni lápices... pero con dodos (). Los usaremos para contar. ¿Cómo?

- Por cantidad: $7 =$ 
- Pero... ¿qué pasa si necesito un número **mayor** a la cantidad de  que tengo? **Agrego más dodos.**
- Es fácil operar (sumamos o restamos ) , pero esta representación no escala. La cantidad de dodos es escasa, más aún en estos tiempos.

Representaciones de números naturales

Asumamos que tenemos  de distinto color. Podemos usar cada color para representar un número distinto ($1 =$ ; $2 =$ ; etc.).

- Conociendo el valor asignado a cada color, podemos operar sin problemas. Pero... ¿qué pasa si se me acaban los colores de ?
- Podemos hacer que los valores dependan de la posición relativa de cada . Por ejemplo:  = 12.

Así nace la **representación posicional**. Desde ahora usaremos la numeración indo-arábica (0-9) para evitar confusiones con los .

Representaciones posicionales

Esta representación se basa en dos elementos:

- Cantidad de símbolos disponibles (base numérica).
- Posición de cada símbolo en una secuencia.

Por ejemplo, así representamos números con tres símbolos:

Secuencia	Valor	Secuencia	Valor	Secuencia	Valor	Secuencia	Valor
0	0	10	3	20	6	100	9
1	1	11	4	21	7	101	10
2	2	12	5	22	8	102	11

Representaciones posicionales

¿Cómo cambia cada secuencia con el incremento de su valor?

1. El símbolo menos significativo (de más a la derecha) **incrementa en una unidad**.
2. Si existe un símbolo para representarlo, se establece.
3. Si no existe un símbolo para representarlo, cambia al primer símbolo de la representación -en general 0- **y se incrementa el siguiente dígito de la secuencia en una unidad**, realizando nuevamente el paso 1 hasta cumplir con la condición del paso 2.

Representaciones posicionales

¿Cómo cambia cada secuencia con el incremento de su valor?

Ejemplo 1: 1021_3

$$1021_3 + 1_3 = 102\mathbf{2}_3 \rightarrow \text{Incrementa su valor sin problema.}$$

$$1022_3 + 1_3 = 10(\mathbf{2+1})\mathbf{0}_3 \rightarrow \text{No puede incrementar su valor. Cambia al primer símbolo de la representación -0- y se incrementa el siguiente símbolo.}$$

$$= 1\mathbf{1}00_3 \rightarrow \text{Incrementa el valor del siguiente símbolo sin problema.}$$

Representaciones posicionales

¿Cómo cambia cada secuencia con el incremento de su valor?

Ejemplo 2: 2222_3

$$2222_3 + 1_3 = 022(2+1)0_3 \rightarrow \text{No puede incrementar su valor.}$$

$$= 02(2+1)00_3 \rightarrow \text{No puede incrementar su valor.}$$

$$= 0(2+1)000_3 \rightarrow \text{No puede incrementar su valor.}$$

$$= 10000_3 \rightarrow \text{Incrementa el valor del siguiente símbolo sin problema, que por defecto es 0.}$$

Representaciones posicionales

Con la siguiente fórmula, podemos obtener el valor de un número de cualquier base numérica en base decimal.

$$\sum_{k=0}^{n-1} s_k \times b^k$$

s = Símbolo (en adelante dígito).

n = Cantidad de dígitos en la secuencia.

b = Base numérica (o número de dígitos).

k = Posición del dígito en la secuencia, siendo 0 la posición del extremo derecho.

Representaciones posicionales - Ejemplo

Supongamos que tenemos tres dígitos como en el ejemplo anterior (base ternaria).

Si tenemos el número 211 en esta base (que será indicada como subíndice en la secuencia), obtenemos su valor de la siguiente forma:

$$(211)_3 = \sum_{k=0}^2 s_k * 3^k = 1 * 3^0 + 1 * 3^1 + 2 * 3^2 = 1 * 1 + 1 * 3 + 2 * 9 = 1 + 3 + 18 = 22$$

Representaciones posicionales

Las representaciones que más utilizaremos a lo largo del curso son dos:

- Binaria, dos dígitos (0 y 1).
 - Notación: Por subíndice o con **b** como sufijo.
 - Ejemplo: 01011**b**
- Hexadecimal, 16 dígitos (0-9, A-F).
 - Notación: Por subíndice, por **h** como sufijo o por prefijo **0x**.
 - Ejemplo: A12**h** / **0x**A12

Representaciones posicionales

¿Por qué son estas las que más utilizaremos?

- Representación binaria será la que nos permitirá operar dentro del computador (circuitos eléctricos, siguiente contenido).
- Representación hexadecimal se usa más en programación: permite reducir la cantidad de símbolos para representar números elevados (ejemplos: RGB, direcciones de memoria, etc.).

Conversión entre base binaria y hexadecimal

Al ser ambas potencias de 2, la conversión entre ambas bases se vuelve trivial:

- **Hexadecimal a binaria**

- Cada dígito de la secuencia se representa en base binaria y luego se concatena el resultado.
- Cada transformación binaria debe representar valores de 0 a 15 (o 16 combinaciones distintas). Como cada dígito otorga dos posibilidades: $2^n = 16 \rightarrow \log_2(2^n) = \log_2(16) \rightarrow \log_2(2^n) = \log_2(2^4) \rightarrow n = 4$

Conversión entre base binaria y hexadecimal

Ejemplo

$$0x9F2 = \left\{ \begin{array}{lcl} 0x9 & = & 1001b \\ 0xF & = & 1111b \\ 0x2 & = & 0010b \end{array} \right\} \rightarrow 0x9F2 = 100111110010b$$

Conversión entre base binaria y hexadecimal

Pequeña “demostración”

$$\begin{aligned} 0x9F2 &= 9 * 16^2 + 15 * 16^1 + 2 * 16^0 \\ &= 1001b * 2^8 + 1111b * 2^4 + 0010b * 2^0 \\ &= (1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0) * 2^8 + (1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0) * 2^4 + (0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0) * 2^0 \\ &= 1 * 2^{11} + 0 * 2^{10} + 0 * 2^9 + 1 * 2^8 + 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 \\ &= 100111110010b \end{aligned}$$

Conversión entre base binaria y hexadecimal

Al ser ambas potencias de 2, la conversión entre ambas bases se vuelve trivial:

- **Binaria a hexadecimal**
 - Se agrupan los dígitos de la secuencia en grupos de 4 y se transforma al dígito hexadecimal que representa el valor.
 - El argumento es el mismo: Cada dígito hexadecimal puede abordar hasta 16 combinaciones distintas de dígitos binarios, esto es, una secuencia de 4 dígitos.

Conversión entre base binaria y hexadecimal

Ejemplo

$$100111110010b = (1001)(1111)(0010) = (0x9)(0xF)(0x2) = 0x9F2$$

Conversión entre base binaria y hexadecimal

Pequeña “demostración”

$$\begin{aligned} 0x9F2 &= 9 * 16^2 + 15 * 16^1 + 2 * 16^0 \\ &= 1001b * 2^8 + 1111b * 2^4 + 0010b * 2^0 \\ &= (1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0) * 2^8 + (1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0) * 2^4 + (0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0) * 2^0 \\ &= 1 * 2^{11} + 0 * 2^{10} + 0 * 2^9 + 1 * 2^8 + 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 \\ &= 100111110010b \end{aligned}$$

Es lo mismo, solo deben leerla de arriba hacia abajo.



Conversión entre base decimal y binaria

Para convertir un número de base decimal a binaria, podemos usar varios métodos. El más común es el **método de divisiones sucesivas**:

1. Se obtiene el resto entre el número en base decimal y el divisor 2.
2. El resto será el **dígito menos significativo del número en base binaria**.
3. Para obtener el siguiente dígito de la secuencia, se realiza la misma operación con el resultado de la **división entera del número en base decimal y 2**.
4. Se termina el procedimiento una vez que la división entera entregue como resultado un 0.

* Este método funciona exactamente igual para transformar números decimales a otras bases.

Conversión entre base decimal y binaria

Ejemplo: 113 en base binaria.

113	% 2 = 1;	113 // 2 = 56	→	$113_2 = \text{XXXXXX}\mathbf{1}$
56	% 2 = 0;	56 // 2 = 28	→	$113_2 = \text{XXXXX}\mathbf{0}1$
28	% 2 = 0;	28 // 2 = 14	→	$113_2 = \text{XXXX}\mathbf{00}1$
14	% 2 = 0;	14 // 2 = 7	→	$113_2 = \text{XXX}\mathbf{100}1$
7	% 2 = 1;	7 // 2 = 3	→	$113_2 = \text{XX}\mathbf{1000}1$
3	% 2 = 1;	3 // 2 = 1	→	$113_2 = \text{X}\mathbf{11000}1$
1	% 2 = 1;	1 // 2 = 0	→	$113_2 = \mathbf{111000}1$

Revisión: $1110001b_{10} = 2^0 + 2^4 + 2^5 + 2^6 = 1 + 16 + 32 + 64 = 113$

Conversión entre base decimal y binaria

¿No hay algún método más rápido? **Sí:**

1. Se obtiene la mayor potencia de 2 que **no sea mayor** al número a convertir.
2. Poner un bit igual a 1 en la posición relativa que corresponda según el valor de la potencia.
3. Repetir los pasos 1 y 2, pero sobre la resta entre el número a convertir y la potencia obtenida.
4. Seguir hasta que la resta sea igual a 0. Las posiciones que no se hayan ocupado tendrán por defecto un bit igual a 0.

Conversión entre base decimal y binaria

Ejemplo: 113 en base binaria.

$$113 - 64 = 49; \quad 64 = 2^6 \quad \rightarrow \quad 113_2 = \mathbf{1}XXXXXX$$

$$49 - 32 = 17; \quad 32 = 2^5 \quad \rightarrow \quad 113_2 = 1\mathbf{1}XXXXX$$

$$17 - 16 = 1; \quad 16 = 2^4 \quad \rightarrow \quad 113_2 = 11\mathbf{1}XXXX$$

$$1 - 1 = 0; \quad 1 = 2^0 \quad \rightarrow \quad 113_2 = 111XXX\mathbf{1}$$

$$(113)_2 = 111\mathbf{000}1b$$

$$\mathbf{Revisión:} \quad 1110001b_{10} = 2^0 + 2^4 + 2^5 + 2^6 = 1 + 16 + 32 + 64 = 113$$

Representaciones de números negativos

Hasta ahora hemos hablado solo de números naturales, es decir, positivos. ¿Qué hay de los enteros, que incluyen a los negativos?

- Existen varias alternativas, con sus pro y contra en términos de representación y operaciones.
- Nos basaremos en la representación binaria para estudiar nuestras alternativas. Ahora, hablaremos directamente de **bits** al hablar de dígitos binarios.



No usaremos dodos negativos. 🙄

Representaciones de números negativos

Opción 1: Agregar un símbolo negativo (-).

- Simple, pero aumenta la cantidad de símbolos que utiliza nuestra representación.
- Veremos más adelante por qué, pero de momento deseamos que nuestro sistema binario se mantenga solo con bits.

Representaciones de números negativos

Opción 2: Agregar un bit de signo (ejemplo: 0 = +, 1 = -).

- Sabemos de forma inmediata qué secuencia es positiva o negativa.
- Tenemos dos representaciones de 0. Con 4 dígitos: 0000 y 1000.
- En términos aritméticos, no nos sirve. Para un número N de nuestra representación, se debe cumplir $N + (-N) = 0$.

N	0101
$-N$	1101
$N + (-N)$	0010

Representaciones de números negativos

Opción 3: Complemento de 1, reemplazamos todos los bits 0 por 1 y vice-versa.

- Nace naturalmente el bit de signo.
- Tenemos una mejora respecto a la aritmética. No obstante, el resultado estará compuesto solo de bits 1.
- Podemos asumir que esta será la representación de 0, pero no es intuitivo.

N	0101
$-N$	1010
$N + (-N)$	1111

Representaciones de números negativos

Opción 4: Complemento de 2, aplicamos complemento de 1 y sumamos una unidad adicional.

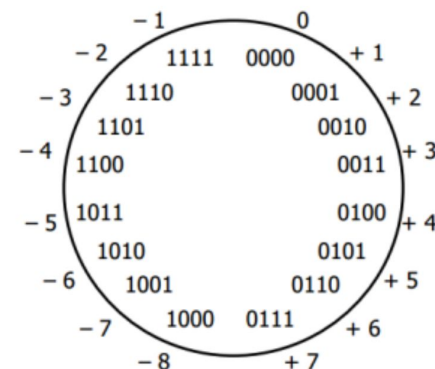
- Sigue naciendo naturalmente el bit de signo.
- Aritméticamente tenemos el resultado deseado: la suma de un número y su negativo entrega solo bits 0. El acarreo final **se descarta**.
- Existen algunos puntos en contra que veremos a continuación, pero se aceptan respecto a lo que ganamos con esta representación.

N	0101
$-N$	1011
$N + (-N)$	0000

Representaciones de números negativos

Contras del complemento de 2

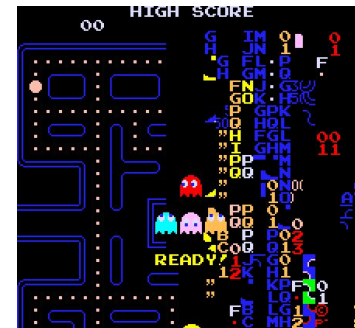
- Representación desbalanceada: como el 0 usa el bit de signo positivo, existe un número positivo que no estará siendo representado.
- **Overflow:** Si una operación aritmética resulta en un valor no representable, nos dará un valor erróneo.
- Ejemplo: $0111b + 0001b = 1000b = -8$, no 8.



Representaciones de números negativos

Ejemplos de *overflow*

- **Pacman™:** El contador de niveles utiliza 8 bits. Al completar el nivel 255, el valor hace *overflow* y cambia a 0. El código del juego no considera este caso y, por ende, realiza acciones no esperadas en pantalla. Más información [aquí](#).
- **Donkey Kong™:** Si bien posee también un caso no controlado en el *timer* del juego, el código verifica que el contador de nivel nunca supere el 100 para evitar *overflow*. Más información [aquí](#).



Representaciones de números negativos

Hack - Cálculo rápido del complemento de 2

1. Leer la secuencia de bits de derecha a izquierda.
2. Al tener la primera ocurrencia de bit igual a 1, dejar todos los bits leídos hasta dicha posición sin modificar.
3. Negar el resto de bits de la secuencia para obtener el resultado final del complemento de 2.

Ejemplo

$$\begin{aligned} 104 &= 01101000b \\ -104 &= C_2(01101000) \\ &= \underbrace{1001}_{\text{Se modifica}} \underbrace{1000}_{\text{Se mantiene sin modificar}}b \end{aligned}$$

* Este método es más rápido para el cómputo manual, pero no para el cómputo digital.

Representaciones de números negativos

Extensión de signo

- Si extendemos la cantidad de bits de un número positivo, se puede ver que solo se extiende su bit de signo igual a 0.

Ejemplo: $8_{2(6 \text{ bits})} = 001000$; $8_{2(8 \text{ bits})} = 00001000$

- ¿Y para números negativos? Con complemento de 2, **funciona de la misma manera.**

Ejemplo: $-8_{2(6 \text{ bits})} = C_2(001000) = 111000$

$-8_{2(8 \text{ bits})} = C_2(00001000) = 11111000$

Representaciones de números negativos

Importante a considerar

- Desde ahora, siempre que trabajemos con números enteros en base binaria, **se asumirá que se usa la representación con complemento de 2.**
- Esto aplica también para **evaluaciones escritas.**
- **¡Evitar el uso de la representación signo-magnitud salvo que se solicite explícitamente!**

Representaciones de números negativos

Ejemplo: Representar -109 en base binaria.

- **Incorrecto:** $(109)_2 = 01101101b \rightarrow (-109)_2 = \mathbf{1}1101101b$
- **Correcto:** $(109)_2 = 01101101b \rightarrow (-109)_2 = C_2(01101101b)$
 $= (10010010 + 1)b$
 $= \mathbf{10010011b}$

Representaciones de números negativos - Dudas adicionales

¿Por qué se llama “complemento de 2”?

- La operación equivale a la diferencia entre la base (en este caso, 2) elevada a la cantidad de dígitos de representación y el número que queremos representar de forma negativa.
- Ejemplo para el complemento de 2 de 7:
 - En resumen: la representación binaria negativa de los números equivale al **complemento de la potencia de 2**.

$$\begin{aligned}C_2(7_{10}) &= \text{binary}(2^4 - 7) \\ &= \text{binary}(16 - 7) \\ &= \text{binary}(9) \\ C_2(0111b) &= 1001b\end{aligned}$$

Representaciones de números negativos - Dudas adicionales

¿Y por qué se llama “complemento de 1” la conversión anterior al complemento de 2?

- Contrario al complemento de 2, en este caso se debe a que la suma entre ambos números entrega solo una secuencia de bits 1.
- Otra forma de verlo es como **el complemento de la secuencia de bits 1**. Ejemplo a continuación.

$$\begin{aligned} 7_{10} &= 1111b + C_1(7_{10}) \\ &= 1111b + 1000b \\ &= 0111b \end{aligned}$$

Representaciones de números negativos - Dudas adicionales

¿Por qué 1000b es -8 para representaciones de 4 bits en complemento de 2?: Podemos usar dos representaciones de 1000b:

Opción 1: Definir dos representaciones para el 0:

$$0000 = +0; 1000 = -0$$

- Respeta el bit de signo, pero perdemos un número representable.
- Aritméticamente, no nos sirve el valor -0:
 $0001b + 1000b = 1001b$; -7 si respetamos la representación de complemento de 2.

Representaciones de números negativos - Dudas adicionales

Opción 2: Asignarle un valor decimal negativo.

- ¿Cuál? El que sea consistente aritméticamente:
 $1 + 1000b = 0001b + 1000b = 1001b = -7$
 $2 + 1000b = 0010b + 1000b = 1010b = -6$
 $3 + 1000b = 0011b + 1000b = 1011b = -5$
- Si extrapolamos a todos los casos, vemos que aritméticamente $1000b$ se comporta como un -8 decimal, siendo ideal esta asociación.

Actividad en clase - Transformación de números decimales

Complete el siguiente código en Python para elaborar una función que transforme un número entero de tipo `int` a `string` binario utilizando la representación de complemento de 2.

```
'''
Actividad en clases: Transformación de base decimal a
base binaria con complemento de 2.
'''

# Función para obtener un número positivo n en binario.
def number_to_bin(n: int) -> str:
    return ''

# Función para aumentar en una unidad un número binario.
def increment_bin_number(n: str) -> str:
    return ''

# Función para obtener un número entero n en binario.
def int_number_to_bin(n: int) -> str:
    return ''

# Resultado: 37 y -37
print(int_number_to_bin(37))
print(int_number_to_bin(-37))
```


Ejercicios

Ahora, veremos algunos ejercicios.

Estos se basan en preguntas de tareas y pruebas de semestres anteriores, por lo que nos servirán de preparación para las evaluaciones.



Ejercicios

Indique la base β en la cual la siguiente ecuación es correcta:

$$7_{\beta} + 8_{\beta} = 13_{\beta}$$

Tarea 1, 2018-2

Ejercicios

Describa el valor decimal del número 0x94A6 si este se interpreta como binario con signo.

Interrogación 1, 2014-2

Ejercicios

Dados $A = 45$ y $B = 57$, ¿cuál es el resultado, en binario, de la operación $A - B$?

Interrogación 1, 2011-2

Ejercicios

Suponga que se tiene un total de 6 bits, usados para representar números positivos y negativos. Dados $A = 27$ y $B = 8$, ¿cuál es el resultado, en binario, de la operación $A + B$? ¿Por qué da este resultado?

Ejercicios

Demuestre que el complemento a 2 del complemento a 2 de un número x es igual a x , esto es: $x = C_2(C_2(x))$

Hint: asuma que $C_2(x + y) = C_2(x) + C_2(y)$

Interrogación 1, 2016-2

Ejercicios

Si hay algún problema eléctrico, como un alza de voltaje, es muy fácil corromper datos almacenados en binario. Por ejemplo, el número 10 (1010b) puede transformarse en 14 (1110b) con tan solo modificar un bit. Describa una codificación binaria para los números 0 y 1, de manera que esta permita detectar y corregir errores de a lo sumo 1 bit, *i.e.*, un bit de la codificación se ve alterado.

Interrogación 1, 2014-2

Ejercicios

En el videojuego The Legend of Zelda™ de la consola Nintendo Entertainment System™, se hace uso de rupias como moneda de cambio. El contador de rupias siempre es mayor o igual a cero, pero cuenta con la particularidad de que deja de incrementarse una vez que llega a 255. Explique, a partir de los contenidos del curso, por qué podría suceder esto.



Examen, 2023-1

Antes de terminar

¿Dudas?

¿Consultas?

¿Inquietudes?

¿Comentarios?





DCC
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 1 - Representaciones Numéricas I

Profesor: Germán Leandro Contreras Sagredo

Anexo - Resolución de ejercicios

¡Importante!

Estos ejercicios pueden tener más de un desarrollo correcto. Las respuestas a continuación no son más que soluciones que **no excluyen** otras alternativas igual de correctas.



Actividad en clase - Respuesta

Complete el siguiente código en Python para elaborar una función que transforme un número entero de tipo `int` a `string` binario utilizando la representación de complemento de 2.

```
# Función para obtener un número positivo n en binario.
def number_to_bin(n: int) -> str:
    n_2 = ''
    # Mientras n sea mayor a 0, dividimos y utilizamos el resto como cada bit
    # de la secuencia.
    while n > 0:
        n_2 = f'{n % 2}{n_2}'
        n //= 2
    if n_2[0] != '0':
        return f'0{n_2}'
    return n_2

# Función para aumentar en una unidad un número binario.
def increment_bin_number(n: str) -> str:
    # Caso base: n == ''. Como ya no quedan bits que incrementar, el bit de
    # acarreo será el bit más significativo de la secuencia y será igual a 1.
    if not n:
        return '1'
    # En otro caso, resolvemos esto con recursión para realizar la suma por
    # acarreo. Si el bit menos significativo es 0, simplemente devolvemos el
    # número con dicho bit igual a 1.
    if n[-1] == '0':
        return f'{n[:-1]}1'
    # Caso recursivo: el bit menos significativo pasa a ser 0 y se incrementa
    # el resto de la secuencia de bits.
    return f'{increment_bin_number(n[:-1])}0'

# Función para obtener un número entero n en binario.
def int_number_to_bin(n: int) -> str:
    # Si el número es positivo, simplemente usamos la función number_to_bin.
    if n >= 0:
        return number_to_bin(n)
    # En otro caso, debemos obtener su complemento de 2.
    n_2 = number_to_bin(-n)
    # Obtenemos el complemento de 1.
    c_1_n_2 = ''.join('1' if bit == '0' else '0' for bit in n_2)
    # Obtenemos el complemento de 2 incrementando la secuencia en una unidad
    # y retornamos el resultado.
    return increment_bin_number(c_1_n_2)

# Resultado: 37 y -37
print(int_number_to_bin(37))
print(int_number_to_bin(-37))
```

Ejercicios - Respuesta

Indique la base β en la cual la siguiente ecuación es correcta:

$$7_{\beta} + 8_{\beta} = 13_{\beta}$$

Respuesta: Para responder esta pregunta se debe considerar que: $13_{\beta} = \beta^1 \cdot 1 + \beta^0 \cdot 3 = 7_{10} + 8_{10} = 15_{10}$.
O sea,

$$\beta + 3 = 15$$

$$\beta = 12$$

Nota: 7 y 8 en base β son iguales a 7 y 8 en base 10, demostrable con la fórmula de transformación decimal.

Ejercicios - Respuesta

Describa el valor decimal del número 0x94A6 si este se interpreta como binario con signo.

Una forma rápida de expresar un número hexadecimal como uno binario, es transformando cada dígito de este a base binaria con 4 dígitos (recordando que $2^4 = 16$):

- $9 = 1001_2$
- $4 = 0100_2$
- $A = 1010_2$
- $6 = 0110_2$

Luego, nuestro número en base binaria corresponde a 1001010010100110_2 . Como este se interpreta como binario con signo, y el bit más significativo corresponde a un 1, utilizamos el complemento a 2 para obtener la representación correcta:

$$1001010010100110_2 = -C_2(1001010010100110_2) = -0110101101011010_2 = -27482$$

Ejercicios - Respuesta

Dados $A = 45$ y $B = 57$, ¿cuál es el resultado, en binario, de la operación $A - B$? Utilice la cantidad mínima de bits necesaria para representar estos números correctamente.

Notemos que si queremos representar estos números en binario, y estamos realizando una operación que implica una resta, entonces necesariamente debemos considerar el bit de signo. Tenemos entonces que $A = 0101101_2$ y $B = 0111001_2$. Luego, restarle a un número positivo uno negativo es equivalente a sumarle su complemento a 2. Entonces:

$$-B = C_2(0111001_2) = 1000111_2$$

$$A - B = 0101101_2 + 1000111_2 = 1110100_2$$

Ahora, este número es negativo, por lo que si queremos su valor decimal correspondiente, realizamos nuevamente el complemento a 2:

$$A - B = -C_2(1110100_2) = -0001100_2 = -12$$

Ejercicios - Respuesta

Suponga que se tiene un total de 6 bits, usados para representar números positivos y negativos. Dados $A = 27$ y $B = 8$, ¿cuál es el resultado, en binario, de la operación $A + B$? ¿Por qué da este resultado?

Si se tiene un total de 6 bits, podemos representar sin problemas los A y B dados como números positivos. Tenemos entonces que $A = 011011_2$ y $B = 000100_2$. Luego, al realizar la operación:

$$A + B = 011011_2 + 000100_2 = 100011_2$$

Podemos ver que el resultado, utilizando representación binaria con signo, es negativo. El número, en base decimal, sería entonces:

$$100011_2 = -C_2(100011_2) = -011101_2 = -29$$

Esto sucede debido a que sobrepasamos nuestro poder de representación. Con 6 bits, el máximo número que podemos representar es $011111_2 = 31$. Si la suma nos da un resultado mayor a ese, al no ser capaces de representar dicho número, obtenemos un número incorrecto (pues la suma de dos números positivos no pueden dar como resultado uno negativo). Este resultado se conoce como **overflow**.

Ejercicios - Respuesta

Demuestre que el complemento a 2 del complemento a 2 de un número x es igual a x , esto es: $x = C_2(C_2(x))$

Hint: asuma que $C_2(x + y) = C_2(x) + C_2(y)$

Solución:

$$x + C_2(x) = 0 \quad / \ C_2$$

$$C_2(x + C_2(x)) = 0 \quad / \text{ utilizamos el hint}$$

$$C_2(x) + C_2(C_2(x)) = 0$$

$$C_2(x) + C_2(C_2(x)) = x + C_2(x)$$

$$C_2(C_2(x)) = x$$

Ejercicios - Respuesta

Si hay algún problema eléctrico, como un alza de voltaje, es muy fácil corromper datos almacenados en binario. Por ejemplo, el número 10 (1010b) puede transformarse en 14 (1110b) con tan solo modificar un bit. Describa una codificación binaria para los números 0 y 1, de manera que esta permita detectar y corregir errores de a lo sumo 1 bit, *i.e.*, un bit de la codificación se ve alterado.

Solución en la siguiente diapositiva.

Ejercicios - Respuesta

Una codificación binaria sencilla para realizar esto, es codificar cada bit como 3 bits de sí mismo, es decir:

$$f(x) = \begin{cases} 111, & x = 1 \\ 000, & x = 0 \end{cases}$$

De esta forma, si tuviéramos el número 0101, este resultaría $f(0101) = 000111000111$. Si uno de los bits se corrompe, basta ir revisando de a 3 dígitos el número, y detenernos cuando no veamos que los 3 bits coinciden para encontrar el espacio que fue corrompido. El bit que predomine corresponderá al número original.

Notar que esto no habría funcionado para la siguiente codificación:

$$g(x) = \begin{cases} 11, & x = 1 \\ 00, & x = 0 \end{cases}$$

Esto, ya que si bien podemos identificar el lugar de corrupción revisando de a 2 dígitos, no podemos saber cuál correspondía al número original (por ejemplo, 10 pudo haber sido 11 o 00, no lo sabemos con dicha codificación).

Ejercicios - Respuesta

En el videojuego The Legend of Zelda™ de la consola Nintendo Entertainment System™, se hace uso de rupias como moneda de cambio. El contador de rupias siempre es mayor o igual a cero, pero cuenta con la particularidad de que deja de incrementarse una vez que llega a 255. Explique, a partir de los contenidos del curso, por qué podría suceder esto.



Solución en la siguiente diapositiva.

Ejercicios - Respuesta

La cantidad de rupias se puede interpretar como un tipo de dato char sin signo ya que su valor varía de 0 a 255, lo que ocurre al representar el dato con 8 bits. Dado esto, el juego controla que no se genere *overflow* ya que en caso de sumar una unidad más a 255 este valor cambiaría a cero.

